VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Databáze pro ukládání a správu časových řad

Databases for Time Series Storage and Maintaining

2016 Bc. Jakub Vašica

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jakub Vašica**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Databáze pro ukládání a správu časových řad
Databases for Time Series Storage and Maintaining

Jazyk vypracování: čeština

Zásady pro vypracování:

V dnešní době vzniká velké množství dat reprezentující různé typy jevů formou časových řad (např. hydrologické měření průtoků, časové řady popisující dopravní tok apod.) a navíc obsahující meta-informace o těchto sekvencích. Pro práci s takovýmito časovými řadami byly vyvinuty speciální typy databází s různou funkcionalitou. Úkolem diplomanta bude prozkoumat různé typy těchto databází a provést jejich srovnání. V další části diplomové práce se bude diplomant zabývat nasazením vybrané databáze a experimenty nad ní. V rámci implementace se také diplomant bude zabývat prací s meta-informacemi o jednotlivých časových řadách.

Jednotlivé body zadání jsou:
1. Analýza existujících databází pro ukládání časových řad a jejich srovnání.
2. Nasazení vybrané databáze.
3. Vytvoření datového modelu ve vybrané databázi.
4. Provedení experimentů s vybranou databází nad experimentálními daty.
5. Vyhodnocení experimentů.

Seznam doporučené odborné literatury:

[1] Jiawei Han and Micheline Kamber "Data Mining: Concepts and Techniques" 2nd ed., Morgan Kaufmann, 2006, Chapter 8: http://web.engr.illinois.edu/~hanj/cs512/bk2chaps/chapter_8.pdf

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Martinovič, Ph.D.**

Datum zadání: 01.09.2015

Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2016 .....

Rád bych na tomto místě poděkoval svému vedoucímu panu Ing. Janu Martinovičovi Ph.D. za rady a konzultace při vytváření této práce.

Abstrakt

Tato diplomová práce se zabývá problematikou databází specializovaných na ukládání a správu časových řad. Nejprve popisuje, co časové řady jsou a jaké jsou možnosti jejich analýzy a uložení. Následně představuje několik databází pro ukládání časových řad a provádí jejich srovnání podle několika hledisek. V poslední části se pak práce věnuje samotnému experimentu. Pro experimentální ověření byly zvoleny InfluxDB, databáze specializovaná na ukládání časových řad, a relační databáze PostgreSQL. K experimentům byla využita vygenerovaná dopravní data a byly připraveny různé testovací scénáře. V závěru je pak provedeno srovnání výsledků těchto různých databázových technologií.

Klíčová slova: časové řady, databáze, InfluxDB, PostgreSQL

Abstract

This thesis addresses the issue of databases for time series storage and maintaining. Firstly, it describes what time series are and how they can be analyzed and stored. Secondly, it presents several databases for time series storage and compares them based on several aspects. Finally, it depicts the experiment. The databases chosen for experimental verification were InfluxDB, a database made especially for time series storage, and PostgreSQL, a relational database. Generated traffic data and different testing scenarios were used for the experiment. In the conclusion, the results of the experiment were presented and compared.

Key Words: time series, databases, InfluxDB, PostgreSQL

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
ARIMA	–	Autoregresive Integrated Moving Average
BI	–	Business Intelligence
CLI	–	Command Line Interface
CPU	–	Central Processing Unit
CSV	–	Comma-Separated Values
GIN	–	Generalized Search Tree
GiST	–	Generalized Inverted Index
GPL	–	General Public License
HDFS	–	Hadoop Distributed File System
HTTP	–	Hypertext Transfer Protocol
IoT	–	Internet of Things
JDBC	–	Java Database Connectivity
JSON	–	JavaScript Object Notation
LGPL	–	Lesser General Public License
LOB	–	Large Object
LSM	–	Log-structured merge-tree
LZO	–	Lempel–Ziv–Oberhumer
MIT	–	Massachusetts Institute of Technology
MVCC	–	Multiversion Concurrency Control
OLAP	–	Online Analytical Processing
ODBC	–	Open Database Connectivity
RAM	–	Random Access Memory
REST	–	Representational State Transfer
RLE	–	Run Length Encoding
SSD	–	Solid-State Drive
SSH	–	Secure Shell
SQL	–	Structured Query Language
TCP	–	Transmission Control Protocol
TMC	–	Traffic Message Channel
TSM	–	Time Structured Merge Tree
TSV	–	Tab-Separated Values
TTL	–	Time To Live
UDP	–	User Datagram Protocol
UTC	–	Coordinated Universal Time
WSGI	–	Web Server Gateway Interface

Seznam obrázků

Seznam tabulek

1 Úvod

Tato diplomová práce se zaměřuje na oblast označovanou jako databáze pro správu a ukládání časových řad. Vzhledem k neustále rostoucímu objemu dat na internetu a rychlému rozvoji nových informačních a komunikačních technologií, jako je například internet věcí, lze předpokládat dynamický rozvoj v této oblasti [50].

Velký objem dat, rozsah časových řad a požadavky na rychlost zpracování kladou další nároky na způsob uložení dat i jejich vyhodnocení. Rozvíjí se oblast specializovaných databází [1]. Cílem této práce je dát přehled o dostupných databázových systémech specializovaných na ukládání časových řad, provést jejich srovnání podle různých hledisek a na vybrané specializované databázi experimentálně prověřit její reálné vlastnosti. Tato práce nejprve v kapitole 2 popisuje, co časové řady jsou a jaké jsou možnosti jejich analýzy a uložení. Specifikuje, jaká data se dají reprezentovat pomocí časových řad, a představuje databáze pro jejich ukládání. Na trhu se v současnosti objevují desítky databází pro ukládání časových řad. Tato práce v kapitole 3 popisuje dvanáct abecedně seřazených databází pro ukládání časových řad. K podrobnému srovnání jsou vybrány čtyři volně dostupné databázové systémy, schopné kromě času a naměřených hodnot ukládat i textová metadata. Srovnání je provedeno v kapitole 4 podle sedmi hledisek. Pro experimentální ověření je zvolena databáze InfluxDB a to z důvodu, že poskytuje uspokojivou množinu dotazovacích funkcí a efektivní vkládání dat a při tom nevyžaduje složitou hardwarovou infrastrukturu – může běžet bez problému na jedné serverové instanci. Pro srovnání výkonu databáze specializované na ukládání časových řad, jsou v diplomové práci přidány testy na databázi PostgreSQL, jejíž počátky spadají do devadesátých let minulého století. Nasazením testovaných databází se zabývá kapitola 5. K experimentům byla využita vygenerovaná dopravní data byly připraveny různé testovací scénáře popsané v kapitole 6. Jak dopadlo srovnání těchto „generačně“ různých databázových technologií a jaké výsledky poskytuje databáze specializována na časové řady v různých situacích, hodnotí kapitola 7.

2 Časové řady

Samotná myšlenka časových řad existuje již dlouho, ale některé metody k jejich ukládání jsou relativně nové. V této kapitole je vysvětleno, co časové řady jsou, jaké známe druhy časových řad, k čemu je možné je využít, jakými metodami je lze analyzovat a jaké jsou možnosti jejich uložení.

Časovou řadou podle Hančlové a Tvrdého (2003) rozumíme „posloupnost hodnot ukazatelů, měřených v určitých časových intervalech. Tyto intervaly jsou zpravidla rovnoměrné (ekvidistantní), a proto je můžeme zapsat následujícím způsobem:

$$x_1, x_2, \ldots, x_n \quad \text{neboli} \quad x_i, i = 1, \ldots, n$$

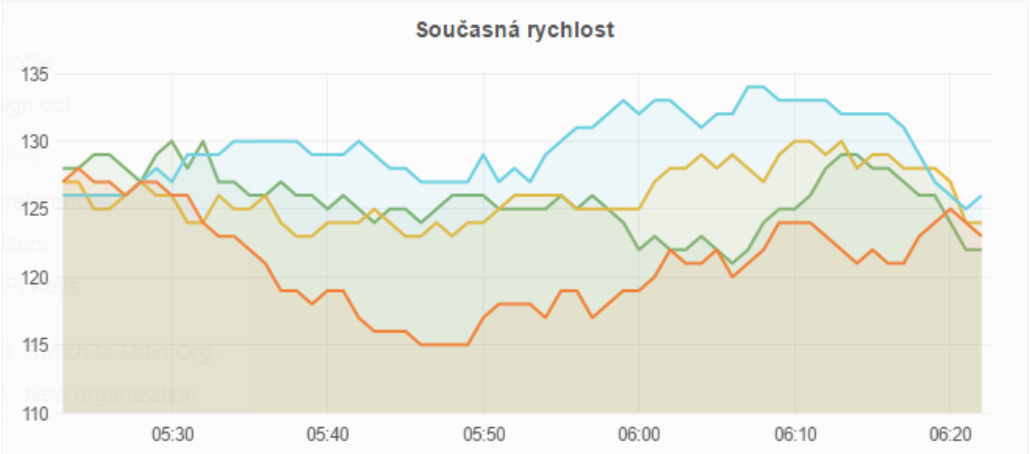
kde x značí pozorovaný ukazatel a i je časová proměnná s celkovým počtem pozorování n"[7]. Časové řady můžeme rozdělit do kategorií z více hledisek, např. podle druhu sledovaných dat, periodicity, charakteru dat aj. Řada může patřit do více kategorií současně, záleží na úhlu pohledu. Časové řady mohou být:

- dlouhodobé – periodicita více než rok,
- krátkodobé – týdenní, měsíční, ...
- intervalové – ukazatel závisí na délce sledovaného intervalu,
- okamžikové – hodnoty vztahující se k určitému okamžiku,
- s absolutními ukazateli – hodnoty ve stejné podobě v jaké byly zaznamenány,
- s relativními ukazateli – hodnoty určitým způsobem transformovány,
- deterministické – konstruovány podle nějakého modelu, neobsahují prvek náhody,
- stochastické – obsahují prvek náhody,
- ekvidistantní – s pravidelným intervalem mezi jednotlivými hodnotami,
- neekvidistantní – s různými intervaly mezi jednotlivými hodnotami.[7]

2.1 Využití časových řad

Existuje celá řada systémů využívajících časové řady. Jedná se např. o systémy pro řízení dopravy, vodních toků nebo monitorování síťového provozu, výkonu počítače, cen akcií a měření teplot, napětí, množství prodejů a mnoho dalších [1]. S rozvojem Internet of Things (IoT) neboli česky internetu věcí se stále zvyšuje množství sbíraných dat ať už z RFID tagů nebo jiných senzorů. Společně s rozvojem technologií pro sběr dat vzniká potřeba vytvářet nové systémy schopné s těmito daty pracovat [5]. Ať už jsou data z jakéhokoliv zdroje, vždy jsou jejich výstupem čísla

chronologicky seřazená podle času měření. Časové řady se vyplatí použít, pokud sbíráme velké množství dat proměnlivých v čase a jednou zaznamenaná data se už nemění [1]. Může se jednat o obrovské objemy dat v řádech terabajtů – desítky gigabajtů denně nebo dokonce desítky gigabajtů dat za minutu. Nashromážděná data slouží k pozdější analýze. Umožňují nám například rozpoznávání trendů nebo pravidelných vzorů v datech a predikci budoucího vývoje. Výstupem z dat mohou být i různé grafy [1]. Příklad vizualizace časové řady rychlostí pomocí grafu je možné vidět na obrázku 1. Získání výsledků v krátkém nebo v lepším případě reálném čase není



Obrázek 1: Vizualizace časových řad z InfluxDB pomocí nástroje Grafana

lehký úkol. Problémy s tím spojené se pokouší řešit řada metod popsaných v dalším textu.

2.2 Metody analýzy časových řad

Časové řady nám poskytují mnoho zajímavých údajů, které nemusí být na první pohled vidět. K jejich získání existuje řada metod a další jsou stále vyvíjeny. Metodu pro analýzu je třeba vybrat na základě konkrétního scénáře – musí se vzít v potaz mimo jiné typ časové řady, účel analýzy a dostupný software a hardware. Nyní si krátce několik metod přiblížíme.

2.2.1 Úpravy časových řad

Dříve než se začne se samotnou analýzou, je zpravidla nutné časové řady modifikovat. Mezi nejčastější úpravy patří:

- časový posun – posunutí celé časové řady v čase o konstantní hodnotu libovolným směrem,
- sezónní diference – umožňuje datům zbavit se sezónního vlivu,
- kumulativní součet – součet hodnot části řady,
- vyhlazování řad – metoda určená k odstraňování odchylek způsobených např. měřením v krátkých intervalech. Hodnoty jsou nahrazovány aritmetickými průměry nejbližších měření, průměry předchozích měření anebo mediánem nejbližších měření.

- doplňení chybějících hodnot – v datech se mohou vyskytovat prázdná místa. Pokud je to z pohledu analýzy žádoucí, lze tyto hodnoty doplnit, a to nulami, podle trendu, aritmetickým průměrem, mediánem, interpolací, extrapolací, aproximací, predikcí nebo jiným adekvátním způsobem [7].

2.2.2 Analýza trendu

Důležitá oblast analýzy časových řad je analýza trendu. Může se jednat např. o lineární růst firmy v průběhu let. Analýza trendu má za cíl především dvě věci, a sice modelovat časové řady (dekompozice) a predikovat budoucí vývoj. Základní čtyři faktory důležité pro analýzu chování u časových řad jsou:

- trendy a dlouhodobé pohyby – výsledek stálého procesu,
- cyklické pohyby nebo cyklické variace - jedná se o dlouhodobé oscilace trendu,
- sezónní pohyby a variace – souvisejí s pravidelnými událostmi v průběhu roku, např. prodeje před Vánoci jsou dvojnásobné oproti prodejům v létě,
- nepravidelné a náhodné pohyby – charakterizují nepravidelné události, např. povodně, stávky atp.

Populární metodou pro modelování časových řad a hledání trendů je regresní analýza, což je jedna ze základních metod statistiky. Před analýzou se často používají některé z úprav zmíněných v části 2.2.1, jako např. vyhlazování řad a sezónní diference. Samotná regresní analýza ovšem není schopna zachytit všechny čtyři výše zmiňované faktory [6].

2.2.3 Predikce dalšího vývoje

Metody predikce dalšího vývoje se snaží nalézt matematickou rovnici, která by přibližně generovala historické vzory v časových řadách. Lze ji poté použít pro dlouhodobé nebo krátkodobé předpovědi budoucích hodnot. Například se můžeme pokusit určit, jaký je předpokládaný prodej v dalším čtvrtletí. Před samotnou predikcí je vhodné nejdříve data vymodelovat a pokusit se odhalit dlouhodobé trendy, což nám může pomoci s výběrem správné metody. Pro předpověď existuje mnoho metod. Metoda používající model Autoregresive Integrated Moving Average (ARIMA), také známá jako Boxova-Jenkinsova metoda, přináší poměrně kvalitní výsledky, ale její použití není jednoduché a kvalita výsledků může záviset na zdatnosti statistika. Pro predikci sezónních vlivů se pak používá modelu SARIMA [6].

2.2.4 Hledání podobnosti

Hledání podobností se snaží nalézt data, která se od dotazované sekvence příliš neliší. Když si vezmeme množinu sekvencí časových řad, máme dvě možnosti hledání podobností, a sice

hledání shody v subsekvencích nebo v celých sekvencích. Hledání shody v celé sekvenci se snaží nalézt množinu navzájem podobných sekvencí, zatímco hledání shody v subsekvencích podobné subsekvence [6].

2.2.4.1 Redukce a transformace dat Vzhledem k obrovskému objemu nasbíraných dat bývá vždy prvním krokem analýzy jejich redukce na menší množství. Zmenšením objemu dat na disku rovněž dojde k rychlejšímu zpracování výsledků. Strategií pro redukci množiny dat je mnoho. Patří mezi ně například výběr podmnožiny atributů, který odstraní nadbytečné atributy či dimenze nebo snížení velikosti dimenze, jež typicky zahrnuje techniky zpracování signálu. Používají se také metody, jež nahrazují data jejich menší reprezentací, např. histogramy, shlukování, vzorkování atp. Mnoho technik pro analýzu signálu vyžaduje, aby data byla ve frekvenční doméně, proto se často používají ortonormální transformace dat [6].

2.2.4.2 Indexování Pro efektivní přístup k datům je vhodné použít multidimenzionální index s pomocí Fourierových koeficientů. Pro hledání shod v subsekvencích se hodí rozdělit každou sekvenci do množiny menších částí. Zrychlit proces hledání podobností se snažila řada indexovacích metod, např. sufixové stromy, R-stromy, R*-stromy a $\epsilon - kdB$ *stromy* [6].

2.2.4.3 Metody pro hledání podobnosti Pro analýzu podobností v subsekvencích časových řad se jako měřítko podobnosti často používá Euklidovská vzdálenost. Čím menší vzdálenost mezi množinami časových řad, tím jsou si více podobné. V některých případech ovšem není možné použít přímo Euklidovskou vzdálenost a je tak třeba se vypořádat s mezerami v datech a rozdíly v měřítku či odchylce časových řad. Přímý způsob, jak vyřešit tyto problémy, je aplikovat normalizační transformaci [6]. Vyhledávání podobností, které zvládá mezery v datech a rozdílné intervaly a odchylky, může vypadat následovně:

1. atomické párování – normalizovat data a najít podobné dvojice malé délky bez mezer,
2. spojování oken – dát dohromady podobná okna a utvořit velké podobné subsekvence,
3. seřazení subsekvencí – seřadit shody subsekvencí a zjistit, zda existuje dostatečné množství podobných částí.

Pomocí tohoto způsobu zpracování dat je možné mezi daty s mezerami a odchylkami různé délky nalézt podobnosti mezi sebou nebo podobnosti vůči dotazu [6].

2.3 Metody ukládání časových řad

Na téma uložení časových řad byla publikována již řada prací. Možností, jak data uložit, je několik. Data mohou být uložena v textovém či binárním souboru ve formě logu, v konvenční relační databázi (např. Oracle, Microsoft SQL Server, ...), v databázi specializované na ukládání časových řad nebo pomocí jiných specializovaných experimentálních metod, které jsou často

součástí akademických a výzkumných prací. V následující části se podíváme na zmíněné způsoby ukládání.

2.3.1 Logy

Nejjednodušší způsob uložení časových řad je pomocí prostých databázových textových souborů ve formě logu. Tento způsob uložení má však řadu nevýhod – velká velikost datového souboru, neefektivní přístup aj. Toto řešení se hodí, pokud pracujeme s malým množstvím časových řad nebo pokud se data vejdou do paměti. Ukázku tohoto způsobu uložení je možné vidět na obrázku 2.

```
2015 7 45.489
2016 1 95.778
2016 2 100.559
2016 1 15.123
```

Obrázek 2: Ukázka uložení dat v prostém databázovém souboru

Pokročilejší metodou je uložit logy v nějakém binárním formátu, což usnadní práci s daty, jelikož se nemusí parsovat. K tomuto účelu se hodí např. formáty jako Apache Thrift[1], Apache Avro[2] nebo Apache Parquet[3]. Tento jednoduchý způsob uložení se může hodit v případě, kdy je množství analyzovaných časových řad relativně malé a rozsahy časových řad jsou velké vzhledem k intervalu vytváření jednotlivých datových souborů. S přibývajícím počtem řad v souboru se ovšem snižuje efektivita dotazování, protože většina dat v souboru není pro dotaz důležitá. Lze použít větší množství souborů, aby počet řad v souboru zůstal nízký. Může se zkrátit interval pro vytváření nového souboru, tím ovšem také zvýšíme počet souborů, což může u některých systémů vést k problémům se stabilitou [5].

2.3.2 Relační databáze

Relační databáze jsou ověřenou a spolehlivou technologií zvládající všemožné scénáře použití. Tradičními relačními databázemi jsou MySQL, PostgreSQL, Microsoft SQL Server, Sybase a Oracle Database. Nejčastější uplatnění nacházejí pro firemní aplikace. Tyto databázové systémy dodržují Structured Query Language (SQL) standard a umožňují provádět transakce plně podle vlastností ACID (Atomicity, Consistency, Isolation, Durability).

Nyní se podíváme na některé možnosti uložení časových řad v relačních databázích, zatím bez ohledu na fyzický návrh, ale pouze schematicky. Data jsou většinou uložena pomocí datového typu **DATE** nebo některé z jeho variant. Některé z proprietráních řešení, jako např. Oracle, IBM Informix nebo Sybase si pro podporu časových řad vytvořily vlastní externí softwarové

[1] https://thrift.apache.org/
[2] https://avro.apache.org/
[3] https://parquet.apache.org/

komponenty. Jednou z možností uložení dat v relačních databázích představuje model navržený Schmidtem (1995), který můžeme vidět v Tabulce 1. Model představuje uložení jedné řady v jedné tabulce, což by pro tisíce a statisíce řad znamenalo vytvoření tisíců a statisíců tabulek. Tato možnost je pro mnoho případů zcela nepoužitelná. Tento model je také v mnoha aspektech svázán se sémantikou konkrétního scénáře a musí se přizpůsobovat individuální aplikaci. Chceme-li přidat nebo smazat řadu, musíme změnit schéma celé databáze [4].

Čas a datum	**Hodnota1**	**Hodnota2**
t1	x1	y1
t2	x2	y2

Tabulka 1: Uložení časové řady v relační databázi [13]

Existují však i další přístupy, které mohou zlepšit využití místa na disku, zvládnout rozdílné časové granularity a zlepšit manipulaci s časovými daty. Další možností, kterou navrhli Lee a Elmasri (1998), je uložit časové řady v externím souboru a z relační databáze na ně odkazovat. Jak je možné vidět v Tabulce 2, první sloupec obsahuje identifikátor řady, druhý název souboru, v němž jsou časové řady uloženy, a třetí kalendář, pod který data patří. Při velkém počtu dat a dotazování bude tento přístup nezvladatelný.

ID řady	**Soubor**	**Kalendář**
id1	nazev1	denně
id2	nazev2	měsíčně

Tabulka 2: Uložení časových řad v relační databázi pomocí externího souboru [10]

Následující způsob navrhli opět Lee a Elmasri (1998). Můžeme uložit jeden typ časové řady do jediné tabulky s tím, že ukládané řady rozlišujeme pomocí jedinečného identifikátoru. Model můžeme vidět v Tabulce 3.

Čas a datum	**Hodnota1**	**Hodnota2**	**ID řady**
t1	x1	y1	id1
t2	x2	y2	id2

Tabulka 3: Uložení časových řad v relační databázi [10]

Jeden řádek představuje jeden záznam, což vede k slabému výkonu a neefektivnímu uložení dat. Opakující se časy generují redundance, což může u většího množství dat vést k řadě problémů. Mimo jiné tento způsob uložení porušuje 3. normální formu. [4]

Podobným způsobem jsou uložena data podle schématu hvězdy (star), tzn. většina dat je ve faktové tabulce a data popisující řadu (metadata) jsou uložena v tabulce dimenzí. Schéma hvězdy může fungovat bez problému až do stovek miliónů dat. [5]

Další modely uložení dat ve své práci zmiňuje Castillejos (2006). Popisuje tři způsoby uložení podle toho, zda je řada univarietní (skládá se pouze z jednoho pozorovaného ukazatele), multivarietní (skládá se z více pozorovaných ukazatelů) anebo nepravidelná. Oproti výše zmíněným modelům tyto modely nevyužívají datový typ DATE. Univarietní řady mohou být uloženy pomocí tří tabulek, které obsahují zaznamenané hodnoty, metadata a popis časových intervalů. Tabulky jsou propojeny pomocí unikátního identifikátoru řady a čísla segmentu. Tabulka s popisem se skládá z počátečního data a konečného data a může rovněž obsahovat i frekvence výskytu. Model tabulky zaznamenaných hodnot je zobrazen v Tabulce 4. Počet sloupců odpovídá počtu hodnot určených v tabulce s popisem časových intervalů. Pokud například provádíme měření každou hodinu, můžeme vytvořit tabulku s 26 sloupci, kde dva sloupce budou sloužit jako identifikátor řady a segmentu a zbylých 24 bude určeno pro zaznamenané hodnoty. Každý sloupec bude obsahovat hodnotu za danou hodinu, a každý řádek tabulky pak hodnoty za jeden den. Uložení multivarietních a nepravidelných dat funguje na podobném principu. Při ukládání multivarietních časových řad přibyde navíc sloupec na určení počtu zaznamenávaných hodnot. [4]

ID řady	**Segment**	**Hodnota1**	**Hodnota2**	**...**	**HodnotaN**
id1	1	x1	x2	...	xN
id1	2	x1	x2	...	xN

Tabulka 4: Model uložení hodnot univarietní řady [4]

Tento přístup přináší úsporu dat na disku, ale zvyšuje náročnost na práci s daty, jelikož se nepoužívá sloupec typu DATE, kvůli čemuž je někdy nutné si funkcionalitu pro práci s daty znovu naprogramovat. Může dojít ke zkomplikování SQL dotazů pro analýzu, např. lze jen velmi těžce provádět agregace a využívat příkaz GROUP BY. Taktéž může být problematické, pokud bychom chtěli změnit velikost intervalu mezi již definovanými hodnotami. [17]

2.3.2.1 Indexování Čtení časových řad bývá sekvenční, proto dává smysl setřídit data podle času a identifikátoru řady, což znamená vytvořit index s těmito sloupci. Když jsou data nesetříděná, přidávání je rychlé, ale dotazování pomalé a naopak. Nejméně efektivní je vyhledávání v případě, kdy není použit žádný index, naopak vytvořením indexu se rychlost vyhledávání dramaticky zrychluje. Pokud je počet záznamů malý, dá se použít technika využívající *signature file*. Od určitého množství záznamů přestane být *signature file* efektivní, potom je lepší použít pro indexování dat stromové struktury, např. B-strom. B-strom je efektivní i pro velké množství časových řad. Volba indexu záleží také na operacích, jaké s daty potřebujeme vykonávat. Pro rychlé vyhledávání subsekvencí je možné použít prostorový index R*-strom. Pro rychlé vyhledávání podobností se hodí R+-strom anebo také R*-strom. [2]

2.3.2.2 Problémy s relačními databázemi V relačních databázích je obtížné změnit interval vzorku dat nebo používat SQL pro analytické dotazy. Mezi další problémy patří analýza

dat s časovými zónami. Data se velmi rychle hromadí. Abychom se mohli dotazovat podle času, musí se nastavit index na atribut s časem. Kvůli tomu rychle narůstá i velikost indexu, která tak snadno může přesáhnout dostupné paměťové možnosti. Způsob uložení neumožňuje příliš dobrou škálovatelnost pro velké objemy dat. [17]

2.3.3 Experimentální způsoby uložení

Jedná se o systémy vzniklé v rámci akademických prací. Snaží se netradičními a novými způsoby o uložení časových řad. Na toto téma byla napsána řada prací, zmíníme si jen některé z nich. Jedním způsobem uložení se zabývá Moudrý (2016) ve své práci *Rozšíření HDF5 o ukládání časových řad.* [11] V pracích *The TS-Tree: Efficient Time Series Search and Retrieval* [3] a *M-tree as an Index Structure for Time Series Data* [15] navrhují autoři novou indexovou strukturu pro rychlé vyhledávání v časových řadách. V práci *A user-defined data type for the storage of time series data allowing efficient similarity screening* [14] navrhli autoři vlastní datový typ na časové řady určený pro databázi PostgreSQL.

2.3.4 NoSQL databáze

Většina databází pro ukládání časových řad je založena na NoSQL. NoSQL je mechanismus umožňující ukládání dat jiným způsobem než v tradičních relačních databázích. Tento přístup přináší mnoho výhod – jednodušší návrh, snadnější zvládání denormalizovaných dat a horizontální škálování serverových uzlů, ovšem na druhou stranu jsou kvůli tomu NoSQL databáze ochuzeny o řadu funkcí, jako jsou transakce, optimalizátor dotazů a další. [17] Výše zmiňovaný způsob uložení dat v relačních databázích podle hvězdicového schématu byl schopen uložit jen jeden záznam na řádek. Apache HBase je schopen na jeden řádek uložit klidně pár stovek tisíc hodnot. NoSQL databáze mohou mít teoreticky neomezený počet sloupců, zatímco relační databáze bývají omezeny v závislosti na databázovém systému v rozsahu desetitisíců a někdy jen tisíců sloupců. NoSQL, jak je z názvu patrné, nevyužívá SQL dotazovací jazyk, ale přesto nezavrhuje klasické SQL. Většina NoSQL databází postrádá vlastnosti ACID (Atomicity, Consistency, Isolation, Durability). Může se stát, že když se více lidí dotazuje na stejná data, uvidí rozdílné výsledky. NoSQL databázím specializovaným na ukládání časových řad se věnují další kapitoly 3 a 4.

3 Databázové systémy pro ukládání časových řad

Označení „databázové systémy pro ukládání časových řad“ není přesně definovaný pojem, ale často tak bývají tyto systémy nazývány. Jedná se o systémy optimalizované pro efektivní sběr, ukládání a dotazování velkého objemu časových řad. Časové řady jsou primárně setříděny podle času a většinou se jejich hodnoty už nemění. Ačkoliv se s těmito daty dá pracovat i v jiných databázových systémech, použití na míru navržených systémů s sebou přináší řadu výhod. Oproti jiným databázovým systémům navíc často podporují následující funkce:

- snížení velikosti vzorku dat – např. GROUP BY TIME (minuta),
- porovnání s dřívějším záznamem – v relačních databázích by to nebyl lehký úkol,
- spojování časových řad – JOIN podle shodných časových razítek [1].

Jejich použití je vhodné u aplikací, kde převažují dotazy podle časového rozsahu. Data se skládají nejméně ze dvojice času a hodnoty. Podobně jako je u dvojice klíč-hodnota (key value pair) základem všeho klíč, tady je základem všeho čas. Podle toho, s jakým databázovým systémem pracujeme, můžeme k datům přidat ještě další popisné informace o datech – tzv. metadata, která slouží k rozlišení dat a filtrování [45].

V současnosti se na trhu pohybují desítky databází pro ukládání časových řad. Zhruba v polovině případů se jedná o proprietární řešení. Některé z nich existují již řadu let a jiné vznikly relativně nedávno. Systémy se výrazně liší kvalitou zpracování, rozsahem nabízených funkcí, fyzickou implementací a prací s metadaty. Některé databáze slouží jen k ukládání čísel. My se dále v kapitole 4 zaměříme na ty databáze, jež umožňují uložit kromě číselných hodnot i metadata. V následující části jsou popsány některé z nejrozšířenějších systémů.

3.1 Axibase TimeSeries Database

Axibase databáze vznikla v roce 2013. Jedná se o NoSQL řešení bez schématu napsané v jazyce Java. Využívá technologie HBase a Hadoop. Podporuje pouze operační systém Linux. Databáze nabízí řadu pokročilých funkcí. Zvládne uložit více typů dat – numerická data i řetězce. K dotazování nabízí SQL podobný jazyk. Umožňuje vkládat data přes protokoly Transmission Control Protocol (TCP), User Datagram Protocol (UDP) a Hypertext Transfer Protocol (HTTP) ve formátu JavaScript Object Notation (JSON), nmon, Comma-separated values (CSV) nebo jako prostý text. Pro komunikaci podporuje mimo jiné Telnet, Java Database Connectivity (JDBC) a HTTP Application Programming Interface (API). Axibase umožňuje integraci se softwarem třetích stran například od firem Amazon, IBM a SolarWinds. Obsahuje řadu zabudovaných portálů pro vizualizaci dat pomocí různých widgetů. Podporuje připojení klientů z jazyků Java, Python, R, PHP a Ruby. Umožňuje nastavit role a oprávnění, definovat triggery a skripty na straně serveru a souběžný přístup. Využívá Shardingu tzn. data jsou rozděleny do samostatných

bloků, což umožňuje horizontální škálování. Shardy mohou být rozprostřeny mezi více serverů. Pro paralelní zpracování používá model MapReduce [48].

K dispozici jsou dvě edice – Comunity Edition a Enterprise Edition. Comunity Edition je zdarma, ale obsahuje pouze omezenou sadu funkcí. Placená edice umožňuje navíc distribuované nasazení, verzování, replikaci dat, kompresi, použití Widgetů a předpovědi pomocí zabudovaných algoritmů (Holt-Winters, ARIMA atp.).

3.2 Blueflood

Uložiště pro časové řady vyvinuté v Rackspace. Na projektu se pracuje od roku 2013. Je dostupný pod licencí Apache 2.0. Podporuje operační systémy Ubuntu a OS X. Je navržen s cílem poskytování služby zákazníkům. Pro fungování vyžaduje nainstalovanou Javu 7 nebo vyšší, Elasticsearch a Cassandru. Systém je více uživatelský a je navržen horizontálně škálovatelný s tím, jak roste počet dat a uživatelů. Podporuje clustery. Skládá se ze tří hlavních modulů - Ingest (vkládání dat), Rollup (agregace) a Query (práce s dotazy). Jako uložiště využívá Cassandru, protože má velkou propustnost zápisu. Cassandra používá sloupcový model a podporuje nativně Time to live (TTL). Data mají vždy pevně danou trvanlivost a po určitém čase mohou vypršet a být odstraněna. Blueflood není určen pro dlouhodobé uchovávání dat. Pro komunikaci využívá HTTP API. POST requesty slouží k odesílání dat, GET requesty k získávání dat. Má zabudovanou experimentální funkci indexování vyhledávání. Data podporují jen pevně danou granularitu a to 5 minut, 20 minut, hodina, 4 hodiny a 24 hodin. Tato skutečnost může být pro spoustu scénářů dost omezující, ale jistě si najde své využití. Databázový systém neumí vykonávat pokročilejší dotazy a nemá žádnou komponentu na vizualizaci dat, ale je schopen pracovat s nástroji jako Grafana[4] a Graphite[5]. Dokumentace na githubu je velmi stručná s chybějícími informacemi [26].

3.3 Druid

Druid je sloupcová distribuovaná databáze napsaná v Javě určená k analytickým výpočtům v reálném čase. První verze vznikla v roce 2012. Druid je volně dostupný pod licencí Apache 2.0. Je schopen zvládnout petabajty dat. Pro svůj chod vyžaduje několik závislostí – Zookeeper, uložiště na metadata (MySQL, PostgreSQL) a uložiště na data (S3, HDFS). Jeho architektura je rozdělena do několika uzlů a podrobněji bude popsána v kapitole 4.Ukládá pouze agregovaná data, jejichž granularitu již nelze změnit. Druid je navržen pro dotazování pomocí metod Business Intelligence (BI) konkrétně pro Online Analytical Processing (OLAP) dotazy. Jazykem pro dotazování je zde JSON, který podporuje dotazy agregační, dotazy na metadata a vyhledávací. Používá kompresi dat a snaží se držet všechna data v paměti. Dotazování a agregace probíhají rychle, protože práce je rozdělena mezi více uzlů. Druid je nejčastěji používán pro analyticky

[4] https://grafana.net/
[5] https://github.com/graphite-project/

zaměřené aplikace používané bežnými uživateli. Je známo, že Druid používaly například služby PayPal a eBay [33] .

3.4 Graphite

Jedná se o nástroj napsaný v Pythonu určený k logování a zobrazování dat. Je dostupný pod licencí Apache 2.0 a pouze pro linuxové operační systémy. Umožňuje uložení numerických dat. Podporuje programovací jazyky Python a JavaScript. Pro své fungování vyžaduje Python 2.7, Django, webový server a Web Server Gateway Interface (WSGI) server. Graphite se dá použít pro následující scénář: uživatel si napíše aplikaci, která se stará o sběr dat, a pak tyto data odesílá do backendové části Graphite označované *carbon*. Data lze poté vizualizovat pomocí webového rozhraní napsaného v Django frameworku [46].

3.5 IBM Informix

Informix TimeSeries je zabudovaná funkce databáze IBM Informix. Administrátoři a vývojáři využivájí Informix TimeSeries k ukládání a analyzování časových řad. Informix umožňuje definovat strukturu dat, řídit kdy a jak jsou data akceptována, řídit datové uložiště, načíst data ze souboru a dotazovat a analyzovat data. Informix ukládá časové řady ve speciálním formátu uvnitř relační databáze, takovým způsobem, aby využil vlastností relačních i nerelačních uložišť. Umožňuje zkombinovat data s informacemi v relačních databázích. Informix data ukládá setříděná podle času a lokality. Databázový server Informix zahrnuje následující funkcionalitu pro správu časových řad:

- speciální datové typy – TimeSeries, Row, Calendar, CalendarPattern,
- TimeSeries SQL rutiny pro dotazování,
- TimeSeries API rutiny a třídy v Javě pro práci s daty.

Informix TimeSeries spolupracuje s dalšími IBM nástroji. Jedná se o IBM® Data Studio nebo IBM Optim™ Development Studio s pomocí nichž se na server nahrávají data. Dále se používá IBM OpenAdmin Tool (OAT) s Informix TimeSeries pluginem pro administraci databázových objektů. Databáze má ovšem jistá omezení – některé SQL příkazy nelze pro časové řady použít. Sloupce s datovým typem TimeSeries neumožňují použití příkazů SELECT UNIQUE, GROUP BY, ORDER BY, FRAGMENT BY, PRIMARY KEY, MERGE, ALTER TYPE a logických operátorů <, <=,=,>,>=. [24]

3.6 InfluxDB

InfluxDB je open source distribuovaná databáze napsaná v jazyce Go. Je k dispozici pro několik linuxových distribucí a Mac OS. Některé starší verze podporují i Windows. Systém je stále vyvíjen – pravidelně každé dva měsíce vychází nová verze. Používání InfluxDB je zcela zdarma

pod licencí Massachusetts Institute of Technology (MIT). Autoři ovšem poskytují, ale i placené služby jako je pronájem serveru, konzultace atp. Oproti většině databází pro práci s časovými řadami InfluxDB nevyžaduje žádné další závislosti. Databáze je bez schématu, lze do ní ukládat libovolná data. Výhodou je, že do ní lze ukládat pravidelné i nepravidelné časové řady, což některé systémy neumožňují. Je určena především pro rychlý zápis a čtení velkého množství dat. Mazání a aktualizace dat jsou možné, ale jedná se o drahé operace. V závislosti na množství ukládaných řad rostou i hardwarové nároky na paměť a procesor. Pro zvýšení propustnosti je zapotřebí navýšit počet jader a gigabajty operační paměti (RAM). S databází můžeme pracovat přes Command Line Interface (CLI), zjednodušené webové rozhraní nebo HTTP API. K dispozici je také několik knihoven pro programovací jazyky Java, Go, .NET, Python a další. Databáze umožňuje použití v clusteru, zálohování a replikaci dat. Je zde možné nastavit politiky pro pravidelné mazání starých dat tzv. Retention Policies. K dotazování je k dispozici SQL podobný jazyk. Podporuje několik agregačních a tranformačních funkcí, funkce pro doplňování chybějících dat a seskupování podle času. Zajímavou funkcí jsou tzv. *Continuous queries*, což jsou dotazy, které jsou vykonávány v pravidelných intervalech a mohou provádět například agregace a snižování vzorku dat. Kromě samotné databáze autoři nabízejí další tři produkty pro sběr a zobrazování dat a pro upozorňování na náhlé změny [25].

3.7 KairosDB

KairosDB je distribuovaná NoSQL databáze postavená na Cassandře. Vychází ze starší verze OpenTSDB. Kromě Cassandry může KairosDB využívat ještě druhé uložiště H2, což je in-memory databáze. Použití H2 se v produkci nedoporučuje. Pro svou činnost vyžaduje nainstalovanou Javu a navýšení počtu deskriptorů souborů. Pro komunikaci s databází se využívá Representational State Transfer (REST) a Telnet API. Z programovacích jazyků nabízí klientskou knihovnu jen pro Javu. Pro vývojářské účely má zabudované webové rozhraní na zobrazování grafů. Oproti OpenTSDB se snaží oddělit získávání dat a jejich prezentaci. Implementuje proto samostatné metody pro agregaci a pro získání dat. Kromě agregací nabízí i metody pro seskupování a filtrování dat podle metadat a času [42].

3.8 KDB+

KDB+ je komerční produkt, který vznikl v roce 2000. Jeho 32-bitová varianta je pro nekomerční využití dostupná zdarma. Jedná se o produkt často používaný ve spojení s finančními službami. Používají jej některé velké investiční banky. KDB+ je napsán v jazyce C a K a podporuje většinu operačních systémů. Kromě toho nabízí API pro C/C++, Javu, .Net, R, Matlab, Perl, Python a také JDBC a Open Database Connectivity (ODBC). KDB používá integrovanou sloupcovou databázi. Část dat si udržuje v paměti pro výpočty v reálném čase, ale jinak jsou všechna data logována do historického uložiště, které zvládne terabajty dat. Podporuje paralelismus a kompresi. Pro dotazování se používá dotazovací jazyk q. Tento jazyk disponuje širokou nabídkou

funkcí – přes 150 příkazů z toho cca 40 matematických funkcí, 7 operací pro spojování tabulek a další operace pro práci s daty a řetězci a jiné systémové příkazy. Operace pro spojování tabulek jsou: Asof JOIN, Equi JOIN, Inner JOIN, Left JOIN, Plus JOIN, Window JOIN. Některé z nich spojují tabulky podobně jako operace JOIN v relačních databázích, ale například Asof JOIN a Window JOIN umí spojit tabulky i na základě definovaného časového úseku [41].

3.9 OpenTSDB

Jedná se o distribuovanou NoSQL databází implementovanou v Javě. Databáze se skládá z více komponent – je založena na Hadoopu a HBase. Podporuje zatím pouze linuxové operační systémy. Počáteční instalace a konfigurace není jednoduchá. Je zapotřebí zprovoznit Hadoop, HBase a ZooKeeper a zajistit, aby mezi sebou jednotlivé části software komunikovaly. OpenTSDB je dostupné pod licencí GNU Lesser General Public License (LGPL) v2.1+, zejména kvůli kompatibilitě s licencemi jeho závislostí. Pro práci s databází se využívá HTTP API a celá řada pluginů. Databáze má přímo v sobě zabudovanou komponentu na generování a zobrazování grafů z dat. Mezi funkce OpenTSDB patří kromě agregací i seskupování, interpolace, snižování množství dat a počítání míry změn v čase.

3.10 Prometheus

Prometheus je systém, napsaný v jazyce Go, který vznikl jako projekt v SoundCloud. Nyní již se však jedná o nezávislý projekt volně dostupný s licencí Apache 2.0. Prometheus je určen pro monitoring a zobrazování upozornění na změny v datech. Upozornění lze zasílat na email, HipChat, Slack, Pushover a další. Umožňuje ukládání pouze numerických dat. Pro uložení využívá multi-dimenzionální model. Časové řady jsou zde identifikovány jménem metriky a dvojicí klíč/hodnota. Celý systém se skládá z několika komponent. Hlavní server Promethea běží samostatně a nevyžaduje žádné další závislosti ani žádnou komplexní infrastrukturu. Každý server je soběstačný, nepotřebuje síťové uložiště ani jakékoliv vzdálené služby. Data jsou v současnosti uložena pouze na lokálním disku. Kromě toho Prometheus experimentálně podporuje ukládání dat do OpenTSDB a InfluxDB. [31]

3.11 RRDTool

RRDTool (Round Robin Database Tool), jehož autorem je Tobi Oetiker, je určený na práci s časovými daty – jejich logování a zobrazování pomocí grafů. RRDTool je napsán v jazyce C a je to volně dostupný nástroj s licencí GNU General Public License (GPL) V2. Velmi často se používá k měření síťového provozu místo staršího Multi Router Traffic Grapher, ze kterého vznikl. Je také zabudován jako doplněk v řadě jiných aplikací. Jedná se o jeden z nejpopulárnějších nástrojů zajišťujících tento typ měření. Jeho první verze pochází již z roku 1999. V současnosti podporuje operační systémy Linux a Windows. Lze v něm uložit pouze numerická data. Podporuje všechny nejrozšířenější programovací jazyky tzn. Java, C, C#, JavaScript, Perl,

Python, Ruby a další. RRD očekává data v pravidelných časových intervalech, které jsou stanoveny již při prvním vytvoření datového souboru. Interval je zde označován jako krok (step) a nelze jej později změnit. Data nemusí vždy přijít přesně ve stejný okamžik a proto RRDTool data automaticky interpoluje, aby odpovídala požadovanému kroku [23].

3.12 RiakTS

RiakTS je distribuovanou NoSQL databází, která byla uvolněna jako open source teprve v lednu 2016. Je optimalizovaná pro časové řady a IoT. Podporuje linuxové operační systémy a Mac OS X. V databázi je možné uložit numerická i textová data v čase s přesností na milisekundy. Pro uložení dat je třeba v databázi navrhnout schéma. Riak používá dotazovací jazyk podobný SQL, ale možností pro dotazování není mnoho – jen několik základních agregačních funkcí a matematických operátorů. Mezi jeho další funkce patří podpora clusterů a automatická replikace dat. Podporuje programovací jazyky Java, Ruby, Python, Erlang a Node.js. Dá se využít např. pro ukládání finančních dat, dat o počasí nebo vědeckých časových řad [32] .

4 Srovnání databází pro ukládání časových řad

V této kapitole se podíváme podrobněji na volně dostupné databázové systémy schopné kromě času a naměřených hodnot ukládat i textová metadata. V další části práce se budeme zabývat testovacím scénářem pracujícím s dopravními daty. Toto bylo zohledněno při výběru databázových systémů. Z nekomerčních řešení byly vybrány databázové systémy Druid, InfluxDB, KairosDB a OpenTSDB. Všechna tato řešení existují na trhu již několik let, jsou stále rozvíjeny a mají aktivní komunitu. Databázové systémy porovnáváme z následujících hledisek:

- architektura databáze,
- vkládání dat,
- votazování,
- práce s metadaty,
- chybějící hodnoty,
- komprese,
- vizualizace dat,

4.1 Druid

Druid běží na systémech Unixového typu tzn. vybrané distribuce Linuxu a Mac OS X. Soustředí se především na rychlost a škálovatelnost. Původně byl navržen čistě jako in-memory databáze a teoreticky jej tak stále lze používat. Protože se jedná o distribuovanou databázi, je možné využít výkon více serverů najednou a zvládne pracovat klidně i s terabajty paměti. Takové řešení může ovšem být finančně náročné, možná také proto autoři časem přidali i možnost perzistentního uložení dat. Minimální doporučované hardwarové požadavky jsou dvě centrální procesorové jednotky (CPU) a 8 GB RAM.

4.1.1 Architektura databáze

Cluster se skládá z více typů uzlů – jak je vidět na Obrázku 3. Uzly mezi sebou komunikují přičemž každý z nich zastává určitou funkci:

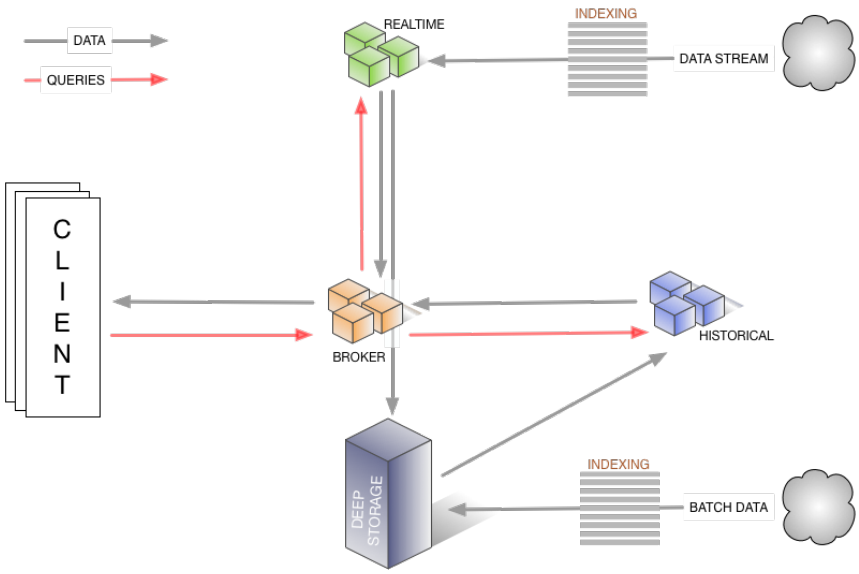
- Historical Nodes – historické uzly tvoří páteř clusteru Druida. Mají lokálně uloženy neměnné segmenty a obsluhují nad nimi dotazy. Uzly se starají o nahrávání, mazání a dotazování segmentů.
- Broker Nodes – na Broker uzly přicházejí dotazy. Tyto uzly jsou zodpovědné za rozdělení dotazu mezi jednotlivé uzly a složení výsledků. Broker má informace o tom, kde je který segment uložen.

- Coordinator Nodes – koordinační uzly spravují segmenty historických uzlů.
- Real-time Processing – zpracování v reálném čase se provádí buď prostřednictvím realtime uzlů nebo indexovací služby. Zpracování zahrnuje vkládání dat, indexování dat (vytvoření segmentů) a předání segmentů historickým uzlům. Data jsou připravena k dotazování ihned po jejich zpracování.[34]

Kromě výše zmíněných uzlů potřebuje systém další tři závislosti:

- ZooKeeper – centralizovaná služba zajišťující údržbu konfiguračních informací a údržbu topologie,
- Metadata Storage – pro uložení informací o metadatech např. Derby, MySQL, PostgreSQL,
- Deep Storage – uložiště velkých objektů (LOB) na uložení dat např. lokální disk, S3, HDFS.[34]

Jak je vidět na Obrázku 3 při vkládání dat, ať už dávkově nebo za běhu, se provádí indexování. Výsledkem indexování je segment. Segmenty jsou základní struktury pro ukládání dat. Ve skutečnosti se segment skládá z více souborů spojených dohromady – kvůli minimalizaci počtu souborových deskriptorů.



Obrázek 3: Architektura databázového systému Druid [34]

4.1.2 Vkládání dat

Druid garantuje spolehlivé vkládání dat v reálném čase skrze veřejné API. Je schopen ukládat pouze agregovaná data podle zadané granularity. V případě konfliktu namísto uložení dat do více řádků dochází k jejich agregaci. Původní data po agregaci už nelze získat zpět a proto je lepší

mít někde jinde uloženou zálohu zdrojových dat. Data se mohou vkládat ve formátu JSON, CSV, Tab-Separated Values (TSV) nebo lze definovat vlastní formát pomocí javascriptového Regex parseru. Při vkládání je nutné definovat schéma.[36] Definice schématu obsahuje následující informace:

- název datového zdroje (tabulky),
- parser – určuje jak data parsovat a obsahuje definici formátu, časového razítka a názvy sloupců,
- seznam agregací,
- granularita – slouží ke stanovení jednotného intervalu mezi segmenty.

Vložení dat vyžaduje jejich indexaci, což systému umožní analyzovat a zkomprimovat data a optimalizovat tím rychlost dotazů. Přímá aktualizace dat není možná. Jedinou možností aktualizace je datový segment smazat a data znovu vložit a zaindexovat. Proces vkládání se skládá z těchto kroků:

1. převod do sloupcového formátu,
2. zaindexování pomocí bitmapových indexů,
3. komprese.

Druid podporuje dva typy vkládání dat: za běhu v reálném čase a dávkové vkládání. Oba typy vkládání lze provést více způsoby.
Vkládání v reálném čase:

- Stream Push – upravíme naši službu, aby odesílala data do Druida pomocí programu Tranquility,
- Stream Pull – žádáme o data z externí služby např. přes Firehose (Kafka a RabbitMQ).

Dávkové vkládání:

- Index Task – vhodný pokud je velikost dávky menší než 1 GB, nepotřebuje žádné externí závislosti,
- Hadoop Task – pro dávky větší než 1 GB, vyžaduje Hadoop cluster.

4.1.3 Dotazování

Druid má vlastní dotazovací jazyk a přijímá dotazy v rámci POST requestů. Všechny uzly obsahující data mají stejné dotazovací API. Dotazy se píšou v JSONu a mají mnoho parametrů, které lze nastavit a použít podle různých scénářů. Typický dotaz obsahuje název datového zdroje,

granularitu výsledných dat, časový rozsah, typ requestu a metriky, které má agregovat. Většina dotazů může obsahovat i množinu filtrů a postagregace (operace prováděné s výsledky agregací, např. aritmetické operace nebo zpracování pomocí zadané JavaScriptové funkce). Když je filtr uveden, bude se skenovat jen odpovídající množina dat.

Zpracování dotazu vypadá následovně: dotaz směřuje nejprve do části označované Broker. Broker porovná dotaz s existujícími datovými segmenty. Poté vybere stroje, které obsahují tyto segmenty a pošle na ně dotaz. Výpočetní uzly dotaz zpracují a vrátí výsledky. Broker obdrží výsledky a sloučí je dohromady, čímž získá finální odpověď, kterou vrátí. Pro filtrování přesnějších výsledků se používají bitmapové indexy uvnitř každého segmentu, díky čemuž výpočetní uzly ví, které řádky odpovídají filtru, aniž by se musely podívat na data. Jakmile uzel zjistí o které řádky se jedná, může přistoupit k požadovaným sloupcům. Pokud dotaz trvá příliš dlouho můžeme jeho vykonávání zastavit. Tato funkce u ostatních porovnávaných databází chybí.

Jsou zde dostupné tři typy dotazů – vyhledávací, agregační a dotazy na metadata. Metadaty se zde myslí systémová metadata, například informace o segmentech, rozsahu dat, datových typech, kardinalitě, názvech sloupců atp. Vyhledávací dotazy vrací hodnoty dimenzí odpovídající specifikaci dotazu. Vyhledávat můžeme shodu v dimenzi s regulárním výrazem anebo s množinou specifikovaných hodnot. Agregační dotazy lze rozdělit do tří skupin – dotazy pomocí časové řady, TopN dotazy a GroupBy dotazy. Dotaz pomocí časové řady vrací pole JSON objektů reprezentující dotazované hodnoty. TopN dotaz vrací množinu setřízených výsledků podle zadaných kritérií. Počet výsledků N je stanoven parametrem *treshold*. TopN dotazy jsou rychlejší než GroupBy dotazy. GroupBy dotaz vrací pole JSON objektů, které tvoří výsledek získaný seskupováním podle specifikovaného dotazu.[35] Mezi dostupné agregační funkce patří:

- COUNT – počet řádků odpovídající zadanému filtru,
- LONGSUM – součet všech hodnot. Výsledkem je 64-bitový integer,
- DOUBLESUM – součet všech hodnot. Výsledkem je 64-bitové desetinné číslo,
- DOUBLEMIN – spočítá minimum ze všech hodnot (včetně kladného nekonečna pro datový typ Double),
- DOUBLEMAX – spočítá maximum ze všech hodnot (včetně záporného nekonečna pro datový typ Double),
- LONGMIN – spočítá minimum ze všech hodnot (včetně maxima pro datový typ Long),
- LONGMAX – spočítá maximum ze všech hodnot (včetně minima pro datový typ Long),
- JavaScriptový agregátor – je zde možnost napsat si vlastní JavaScriptové agregační funkce,
- HYPERLOGLOG – výpočet odhadované kardinality.[21]

Nelze provádět operaci JOIN, je to možné jen v omezené míře v rámci experimentální operace Lookup.

4.1.4 Práce s metadaty

Metadata jsou ukládána společně s ostatními daty. Druid používá slovníkové kódování, takže textové řetězce nahrazuje při kompresi číselnou hodnotou. Metadata lze využít u dotazování pro filtrování výsledků.

4.1.5 Chybějící hodnoty

Chybějící hodnoty lze v dotazech nahradit v rámci transformačních funkcí a u operace Lookup můžeme specifikovat, zda chybějící hodnoty zachovat, vynechat anebo nahradit jinou hodnotou.

4.1.6 Komprese

Kompresní metoda je vybírána podle datového typu sloupce. Používané kompresní metody jsou:

- LZ4[6] – na všechny sloupce,
- slovníkové kódování – na sloupce s textovými řetězci,
- bitmapová komprese – na bitmapové indexy[7]. [34]

4.1.7 Vizualizace dat

Data lze vizualizovat pomocí webového uživatelského rozhraní Pivot[8], které je založeno na JavaScriptové knihovně Plywood. Další vizualizační nástroj kompatibilní s Druidem se nazývá Caravel[9]. Umožňuje z množiny dat vytvářet a sdílet interaktivní dashboardy. Jednoduché dashboardy umí vytvářet také Metabase. Nebo lze pro vizualizaci využít univerzální nástroj Grafana.

4.2 InfluxDB

Databázový systém InfluxDB nevyžaduje žádné závislosti (Hadoop, Redis, Cassandra, HDFS, ZooKeeper . . .). Umožňuje jednoduché nasazení – stačí ho jen nainstalovat a spustit. Hardwarové nároky závisí na kardinalitě dat, na množství dat a také na tom kolik očekáváme zápisů a dotazů za sekundu. Obecně se doporučuje pro zrychlení dotazů použít Solid-State Drive (SSD) disky a větší množství paměti (RAM).

4.2.1 Architektura databáze

InfluxDB se skládá ze dvou databází – jedna je na uživatelská data, druhá na interní informace o systému, o clusterech atp. Databáze je zcela bez schématu. Lze libovolně vkládat data a není

[6] http://cyan4973.github.io/lz4/
[7] http://www.oracle.com/technetwork/articles/sharma-indexes-093638.html
[8] https://github.com/implydata/pivot
[9] http://airbnb.io/caravel/

problém za běhu přidat další sloupce, ale datové typy sloupců musí zůstat konzistentní. InfluxDB implementuje vlastní zabudované uložiště dat. Ve svém fyzickém návrhu využívá datovou strukturu nazývanou Time Structured Merge Tree (TSM), je v některých věcech podobná datové struktuře Log Structured Merge Tree (LSM) [30] , kterou používá např. Apache Cassandra, HBase, LevelDB a Riak. TSM se skládá ze tří částí - Write Ahead Log (WAL), indexované datové soubory a několik dalších souborů, ve kterých jsou komprimovaná metadata. Doporučuje se mít adresář pro WAL a pro data na rozdílných discích. Data jsou organizovaná do tzv. Shardů, což jsou neměnné souvislé bloky času. Bloky dat jsou v datových souborech seřazeny podle klíče a podle času, což umožňuje efektivní RANGE SCAN. Klíč je tvořen z názvu měření a metadat. Jednou za čas jsou datové soubory zkombinovány neboli stlačeny (*compaction*). Stlačení provede spojení těchto bloků do větších bloků času, díky čemuž se zlepšuje komprese. Datové soubory se navzájem nepřekrývají, což umožňuje snadné smazání starých dat. Metadata (informace o databázích, metrikách, názvech a datových typech) jsou načtena do paměti při spuštění databáze, a pak už se k nim na disku přistupuje jen když jsou modifikována [27]. InfluxDB používá nepřímé indexování pomocí namapování souborů do paměti tzv. memory mapped file (mmap). Datové soubory mají indexy na začátcích bloků a při spuštění databáze se všechny procházejí a mapují do paměti. Mmap zajistí, že se operační systém bude starat o cachování. Zabere sice hodně paměti, ale pokud na serveru neběží nic jiného než databáze, tak by to neměl být problém. Samotný obsah souborů není načten ve fyzické paměti, jsou na něj pouze udržovány odkazy a k obsahu se přistupuje až poté co je vyžadován [19].

Datový model InfluxDB se skládá z měření (měření jsou podobná tabulkám), kterých mohou být až tisíce. Uvnitř měření je množina oddělených časových řad, které jsou definovány množinou tagů. Tagy jsou dvojice (klíč-hodnota) reprezentující metadata. Všechny tagy jsou automaticky indexovány. Každé měření může obsahovat teoreticky neomezený počet polí (sloupců) s hodnotami – lze tedy ukládat multivarietní časové řady. Pole s hodnotami se neindexují. Všechno je indexováno podle času a podle řad. InfluxDB podporuje následující datové typy: boolean, integer, float, string, timestamp (unixový čas), a to až v nanosekundách.

4.2.2 Vkládání dat

Data lze vkládat přes HTTP API, CLI, UDP, pluginy anebo přes zjednodušené webové rozhraní. Zdrojem dat může být i jiná databáze nebo služba, například OpenTSDB, Graphite nebo collectd. Data lze vkládat jednotlivě nebo po malých dávkách – cca v řádech tisíců. Doporučovaná velikost dávky je 5000 záznamů. Hlavní metodou transportu je HTTP API. Přenos přes UDP je třeba explicitně povolit v konfiguračním souboru a čas vkládaných dat musí být uveden v sekundách. Nelze nastavit jinou přesnost. V případě, kdy potřebujeme data s jinou přesností než v sekundách, je nutné použít pro vkládání HTTP API.

Při vkládání dat se data zapíšou nejprve do Write Ahead Logu a hned poté do Cache, aby se okamžitě dala dotazovat. Z Write Ahead Logu jsou data pravidelně zapisována do datových

souborů pomocí operace *flush*. Zápis do WAL je velmi efektivní, protože slouží pouze k přidávání dat [19].

4.2.3 Dotazování

Influx Query Language (InfluxQL) je dotazovací jazyk SQL. Používá se podobným způsobem jako SQL v kterémkoliv jiném prostředí a zároveň umožňuje práci s časovými řadami. Dotazy je možné posílat přes CLI nebo posíláním GET requestů na HTTP API na koncový bod */query*. Klientské knihovny využívající HTTP API jsou implementovány pro následující programovací jazyky: Erlang, Go, Haskell, Java, JavaScript, Lisp, .NET, Perl, PHP, R, Ruby a několik dalších. InfluxDB nepodporuje logické operátory a operátory pro porovnávání. Pro agregování hodnot lze použít tyto funkce: COUNT, MEAN, DISTINCT, SUM, BOTTOM, TOP, MAX a MIN. Další méně rozšířené agregační funkce jsou:

- SPREAD – rozdíl mexi minimální a maximální hodnotou vybraného pole,
- MEDIAN – střední hodnota,
- FIRST – nejstarší hodnota,
- LAST – nejnovější hodnota,
- DERIVATIVE – míra změny hodnot pole,
- NON_NEGATIVE_DERIVATIVE – nezáporná míra změny hodnot pole,
- PERCENTIL – percentil,
- STDDEV – směrodatná odchylka.[25]

Když na databázi přijde dotaz a je spojen se specifickým klíčem a polem, nejprve proběhne binární vyhledávání v datových souborech, které odpovídají zadanému intervalu. Pak se musí najít pozice bloku dat v rámci souboru. Podle hashe se zjistí identifikátor záznamu. Následně se podle identifikátoru a celkového počtu záznamů binárním vyhledáváním v indexu v paměti přesně zjistí počáteční pozice bloku dat.[19] Výsledky dotazů se vrací ve formátu JSON. Výsledný JSON má minimální podobu. Pokud je však flag *pretty* nastaven na hodnotu *true* je JSON naformátovaný v čitelné podobě, což se hodí pro debugování, ale v produkci to zbytečně zvyšuje síťový přenos. Všechny časy v InfluxDB používají koordinovaný světový čas (UTC) – to platí i pro dotazování. Pokud dotaz vrací velké množství dat, jsou výsledky vráceny po menších dávkách, např. po 10000 záznamech. Velikost dávky je možné explicitně nastavit parametrem *chunk_size* [25].

4.2.4 Práce s metadaty

Metadata jsou zde označována jako tagy. V dřívějších verzích InfluxDB tagy nebyly k dispozici, ale na žádost komunity je autoři časem přidali. Jsou udržovány v paměti a indexovány. Jejich použitím ve WHERE podmínce zrychlíme dotazy. Metadata jsou uložena v samostatném souboru serializované ve formátu JSON a navíc zkomprimované pomocí Snappy[10] komprese. Při zápisu nových metadat z Write Ahead Logu je tento soubor načten, dekomprimován, rozšířen o nový záznam a poté je zapsán jako nový soubor.

4.2.5 Chybějící hodnoty

Pro intervaly bez dat vrací InfluxDB defaultně hodnotu *null*. Pomocí funkce FILL je možné tuto hodnotu změnit. Funkce FILL může přijímat parametry *null*, *none*, *previous* nebo libovolné číslo, může nahradit hodnoty nullem, vynechat je z dotazu, nahradit je hodnotou z předcházejícího intervalu anebo je nahradit zadaným číslem [25].

4.2.6 Komprese

InfluxDB defaultně komprimuje všechny bloky dat, aby se snížilo množství přístupů na disk během dotazování. Časová razítka a hodnoty jsou uloženy separátně jako dvě komprimované části. Jsou použity různé druhy komprese podle datového typu a přesnosti času. Je rozdíl jestli se ukládá čas s přesností na sekundy nebo nanosekundy. Záleží také, jak velké jsou rozestupy mezi jednotlivými časy. Podle toho se používá buď simple8b[11], delta komprese[12] nebo Run Length Encoding (RLE)[13], v některých případech jsou časy uloženy zcela bez komprese. Datový typ Float je komprimován pomocí implementace delta kódování převzaté z databáze Gorilla[14] od společnosti Facebook. Datový typ Boolean je komprimován pomocí bitů. U Integerů záleží na jejich rozsahu a je použita buď kombinace zig zag kódování a simple8b nebo jsou uloženy bez komprese. Na řetězce se zde používá komprese Snappy [27]. Nastavení komprese není možné změnit, probíhá zcela automaticky. Můžeme ji ale ovlivnit nastavením maximálního počtu bodů v datovém bloku. Vyšší čísla znamenají lepší kompresi, ale zase zpomalí dotazování.

4.2.7 Vizualizace dat

Vizualizovat data z databáze lze pomocí nástrojů Chronograf[15] a Grafana. Pomocí obou nástrojů lze vytvářet přehledné dashboardy. Chronograf je jednoduchý grafovací nástroj pocházející přímo od tvůrců InfluxDB. Zatím ale neobsahuje příliš mnoho funkcí a je stejně jako InfluxDB stále rozvíjen. Grafana je propracovanější než Chronograf. Podporuje i jiné zdroje dat než InfluxDB

[10] https://github.com/google/snappy
[11] https://github.com/jwilder/encoding/tree/master/simple8b
[12] http://www.dspguide.com/ch27/4.htm
[13] http://www.cs.vsb.cz/benes/vyuka/pte/texty/komprese/ch02s01.html
[14] http://www.vldb.org/pvldb/vol8/p1816-teller.pdf
[15] https://influxdata.com/time-series-platform/chronograf/

a lze ji rozšiřovat pomocí doplňků. Ukázka vizualizace dat z InfluxDB pomocí nástoje Grafana je vidět na obrázku 1.

4.3 KairosDB

Databáze vznikla z alternativní větve OpenTSDB. V některých věcech se OpenTSDB stále podobá, ale je zde několik zásadních odlišností, např. KairosDB neprovádí interpolaci dat, používá jiné uložiště a je zde odděleno získávání dat a jejich zobrazování – KairosDB generuje grafy na klientovi zatímco OpenTSDB na serveru. KairosDB defaultně obsahuje „in memory“ databázi H2. Pro vývoj je to dostačující, protože se nemusí instalovat a konfigurovat Cassandra, ale H2 může být ve výsledku pomalejší. Místo H2 se proto v produkci používá Cassandra.

4.3.1 Architektura databáze

Cassandra defaultně používá široké řádky. Každý řádek má šířku nastavenou na tři týdny. Důvodem je, že Cassandra má limit 2 miliardy sloupců na řádek, což odpovídá cca třem týdnům. Čím více dat se vleze na řádek, tím lépe půjde dotazování. Po zaplnění řádku Cassandra jednoduše začne zapisovat na řádek nový. Oproti OpenTSDB se zde nepoužívají unikátní identifikátory jako odkaz na metadata. Cassandra zapisuje metadata přímo do řádkového klíče (row key) a patřičných indexů [49] a využívá LSM stromy pro zrychlení zápisu na úkor pomalejšího čtení [30]. Schéma Cassandry se skládá ze tří rodin sloupců:

- Data Points – zde jsou uložena data. Klíč řady je tvořen kombinací názvu metriky, časového razítka (začátku řady), typu uložiště a řetězce tagů.
- Row Key Index – index určený k vyhledávání řad během dotazu. Klíčem řady je název metriky a názvy sloupců jsou klíče řad z rodiny Data Points. Sloupce neobsahují žádnou hodnotu.
- String Index – určen k odpovídání na dotazy – na metriky, tagy a hodnoty tagů v systému. Celkem má tedy tři řádky [49].

Datové typy v KairosDB jsou: integer, float a textová metadata. Je možné definovat si vlastní datový typ. Čas musí být vždy v milisekundách a musí se jednat o Unixový čas.

4.3.2 Vkládání dat

Vkládání dat je možné buď přes Telnet, HTTP REST API nebo pomocí knihoven třetích stran. Příkaz je specifikován pomocí JSONu. JSON s daty můžeme před odesláním zkomprimovat pomocí gzip. V rámci HTTP requestu může být v JSONu posílán jeden nebo více datových bodů. Datové body se vztahují pouze k jedné veličině (metrice). Metrika může mít v JSONu jeden nebo více datových bodů zároveň. Pokud je třeba ukládat více typů hodnot, je nutné každou vkládat jako jinou metriku. Aktualizovat záznam lze jedině vložením záznamu se stejným

názvem metriky, stejnými tagy, stejným časem a stejným datovým typem. Původní záznam se tak přepíše. Data lze smazat posláním requestu s dotazem v těle na koncový bod */delete*. Dojde ke smazání všech datových bodů, které dotaz vrátí.[38]

4.3.3 Dotazování

KairosDB disponuje vlastním dotazovacím jazykem. Dotazy se posílají pomocí REST API ve formě JSONu. V dotazu se musí specifikovat název dotazované metriky a počáteční čas a může se specifikovat i koncový čas (není povinný). Čas se uvádí buď absolutně anebo relativně. Absolutní čas se uvádí v milisekundách. Relativní čas se uvádí vzhledem k současnému času a je možné jej uvést v různých jednotkách – milisekundy, sekundy, minuty, hodiny, dny, týdny, měsíce, roky. Při dotazování nelze použít operaci JOIN [40].

Když přijde dotaz na měření (metriku) přečte se row key index, což vrátí řádky obsahující požadovaná data. Řádkové klíče jsou poté filtrovány na základě tagů (metadat). Následně se pošle několik volání na klienta Cassandry (hector), aby načetl data z více řad. Pokud některé z řad obsahují příliš mnoho dat, jsou nahrány jednotlivě pomocí velkébo bufferu [49].

Výsledky dotazu je možné seskupovat třemi způsoby – podle tagů, časového rozsahu nebo podle hodnot. Dále je možné v dotazech použít agregace. Agregátory je možné kombinovat. Jsou zpracovávány podle pořadí, v jakém byly zapsány. Data lze také filtrovat pomocí filtrů. Filtrování se provádí na základě uvedené množiny tagů. Dotaz vrátí jen takové body, které jsou spojené s daným tagem. Pro dotazování jsou k dispozici následující agregační funkce: AVERAGE, MIN, MAX, COUNT, SUM, FIRST, LAST. Další speciální agregační funkce jsou:

- DEV – směrodatná odchylka,
- PERCENTILE – percentil,
- GAPS – nahrazení chybějících hodnot,
- LEAST_SQUARES – metoda nejmenších čtverců,
- DIFF – rozdíl po sobě následujících datových bodů,
- RATE – míra změny za jednotku času,
- TRIM – odstranění počátečních nebo koncových bodů,
- SCALE – změna hodnot v zadaném měřítku,
- SAVE_AS – uložení výsledků do jiné metriky [40].

4.3.4 Práce s metadaty

Metadata se zde nazývají tagy. Umožňují zůžit výsledky dotazů – používají se při filtrování. Všechny záznamy musí mít vždy uveden minimálně jeden tag.

4.3.5 Chybějící hodnoty

Agregátor GAPS nahrazuje chybějící hodnoty hodnotou *null*. Jiné možnosti doplnění nebo nahrazení chybějících hodnot jako interpolace, extrapolace atp. KairosDB nenabízí.

4.3.6 Komprese

Cassandra umožňuje nastavit pro tabulku kompresi pomocí LZ4, Snappy nebo Deflate. Defaultně je v Cassanře použita komprese LZ4 [22].

4.3.7 Vizualizace dat

KairosDB nemá vlastní nástroj na vizualizaci dat, ale existují možnosti integrace do následujících nástrojů:

- Grafana – existuje KairosDB plugin pro Grafanu,
- facette.io[16] – nástroj určený pro vizualizaci časových řad,
- Cubism.js[17] – D3 plugin pro vizualizaci časových řad,
- Keen Dashboard Templates[18] – nástroj na tvorbu responzivních dashboardů pomocí Bootstrap,
- Dygraphs[19] – JavaScriptová open source knihovna pro tvorbu grafů.

4.4 OpenTSDB

OpenTSDB je škálovatelná, distribuovaná databáze. Hodí se jako dlouhodobé uložiště dat. Vždy zachovává data s původní přesností a umožňuje jejich analýzu. Je postavena nad uložištěm Apache HBase. Schéma uložení v HBase je optimalizované pro minimální velikost na disku a rychlé agregace. Kromě HBase pro svůj běh potřebuje nainstalovanou Javu, ZooKeeper a GnuPlot.

4.4.1 Architektura databáze

Architektura OpenTSDB je znázorněna na Obrázku 4. Jak je vidět OpenTSDB se zkládá z Time Series Daemonů (TSD) a množiny CLI utilit. TSD jsou na sobě nezávislé a může jich běžet libovolný počet pro zvládnutí požadované zátěže. Pro ukládání dat používá TSD uložiště HBase. Zpočátku se v OpenTSDB používaly jen dvě tabulky. Jedna velká se všemi datovými

[16] https://facette.io/
[17] https://square.github.io/cubism/
[18] https://github.com/keen/dashboards
[19] http://dygraphs.com/

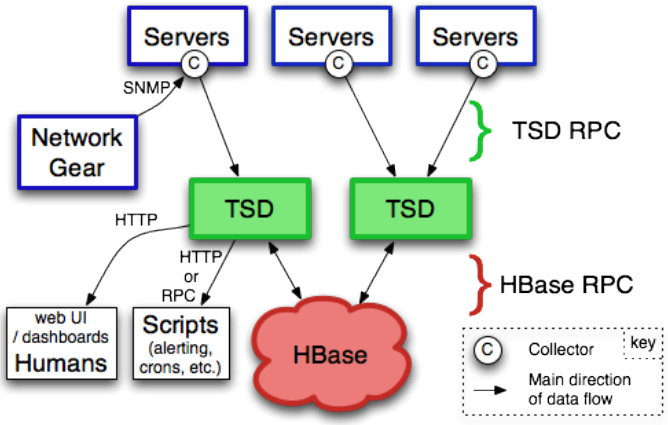
body nazvaná *tsdb* a druhá menší *tsdb-uid* na metadata. Od verze 2.0 přibyly ještě tabulky *tsdb-meta* a *tsdb-tree*, aby se lépe poznalo, kde se časové řady nacházejí a bylo je možné hierarchicky organizovat.[28] Tabulka tsdb se skládá z následujících částí:

- row key – klíč řádku je složen z názvu metriky, časového razítka, množiny názvů tagů a jejich hodnot
- column qualifier – určuje datový typ sloupce
- column value – hodnota sloupce

Jeden řádek tabulky obsahuje data za hodinu, což znamená 60 sloupců pro data s granularitou po minutě, 3600 sloupců s granularitou po sekundách atd.

Menší tabulka tsdb-uid s metadaty a názvy metrik používá obousměrné mapování mezi jmény a unikátními identifikátory. Obsahuje dvě rodiny sloupců:

- name – mapuje unikátní identifikátor na řetězec
- id – mapuje řetězec na unikátní identifikátor [28].



Obrázek 4: Architektura databázového systému OpenTSDB [37]

Datové typy v OpenTSDB jsou integer, float, string (metadata) a timestamp (v milisekundách v unixovém čase).

4.4.2 Vkládání dat

Není třeba vytvářet žádné schéma. Data se jednoduše vloží a to přes telnet, HTTP API, pomocí knihoven třetích stran nebo hromadným vkládáním po dávkách přes CLI pomocí utility *import*. OpenTSDB neumožňuje vkládat více stejných datových bodů. Pokud se stane, že bychom vložili dvakrát stejný datový bod např. provedli dvakrát stejný insert, data se sice uloží, ale při dotazování vyhodí databáze vyjímku a upozorní na duplikát v datech. Abychom se chyby zbavili musíme konflikt sami vyřešit nebo nastavit v konfiguračním souboru, aby se konflikty řešily automaticky. Databáze poté vrací novější hodnotu [20].

4.4.3 Dotazování

Uložené časové řady je možné dotazovat a vykreslit pomocí zabudovaného webového rozhraní. Kromě toho je pro dotazování k dispozici HTTP API, CLI nástroj nebo můžeme dotazy posílat pomocí QueryString requestů. Součástí každého dotazu musí být alespoň jedna agregační funkce, název metriky a počáteční čas. OpenTSDB neumí operaci JOIN. Z běžných agregačních funkcí podporuje: AVG, COUNT, MAX, MIN a SUM. Další speciální agregační funkce jsou:

- DEV – směrodatná odchylka,
- PERCENTILES – výpočet percentilů,
- NONE – vynechání agregace. Používá se pro získání čistých dat,
- MIMMIN – vrací maximální možnou hodnotu datového typu pokud chybí minimum,
- MIMMAX – vrací minimální možnou hodnotu datového typu pokud chybí maximum,
- ZIMSUM – funkce provádí SUM všech hodnot, ale neprovádí interpolaci – místo toho nahradí chybějící hodnoty nulami [39].

4.4.4 Práce s metadaty

Metadata se zde nazývají tagy. Tagy jsou sdíleny více unikátními časovými řadami a umožňují tak rozlišit související záznamy. Často se používají pro označení zařízení, ze kterého data pocházejí. Jejich použití urychluje dotazování a umožňuje velmi rychlé agregace. Oproti KairosDB nejsou uloženy přímo s daty, ale ve zvláštní tabulce a jsou nahrazeny unikátním identifikátorem.

4.4.5 Chybějící hodnoty

K doplňění chybějících hodnot dochází při agregování. Může se stát, že se vyskytnou mezery v datech nebo jsou data zpožděná a při agregování se jejich časy nepřekrývají. Můžeme taková data ignorovat nebo nahradit nějakou skalární hodnotou. Chybějící hodnoty mohou být nahrazeny také hodnotami *Null* nebo *NaN*. Někdy ovšem potřebujeme, aby se hodnoty inkrementálně navyšovaly a v takových případech se vyplatí přibližně odhadnout vývoj řady pomocí interpolace. OpenTSDB implementuje lineární interpolaci. Pokud není specifikováno jinak, provádí se interpolace automaticky a to vždy, když se při agregaci vyskytují prázdné hodnoty nebo časové intervaly neodpovídají. Výsledky pak mohou být zkreslené a je třeba si na to dávat pozor [39].

4.4.6 Komprese

Komprese je defaultně vypnutá, ale při použití v produkci se ji doporučuje zapnout. Metoda komprese se volí při vytváření tabulek v HBase. Lze zvolit Lempel–Ziv–Oberhumer (LZO), gzip nebo Snappy. Autoři na stránkách uvádějí, že s kompresí LZO lze v průměru dosáhnout

kompresního poměru 4,2 na tabulce. Výsledky se samozřejmě mohou lišit podle konkrétních dat [29].

4.4.7 Vizualizace dat

OpenTSDB má zabudouvanou komponentu GnuPlot[20] pro vizualizaci dat prostřednictvím webového rozhraní. Vykreslený graf zobrazuje více dat než, které má vracet zadaný dotaz. OpenTSDB se snaží vykreslit smysluplné grafy, aby správně vykreslil hodnoty na koncích grafu, přečte z databáze větší rozsah dat. Velikost dat, které čte navíc, je úměrná k zadanému časovému úseku. Pro základní použití sice GnuPlot vystačí, ale existují i propracovanější nástroje. Komunita vytvořila pro vizualizaci dat z OpenTSDB například Metrilyx[21], Status Wolf[22] nebo TSDash[23]. Lidé ze StackExchange vyvinuli monitorovací framework Bosun[24]. Existuje také OpenTSDB doplněk pro Grafanu.

4.5 Srovnání

Srovnání některých vlastností databází je vidět v Tabulce 5. Nelze jednoznačně tvrdit, že některá z databází je lepší nebo horší. V mnoha ohledech jsou si databáze podobné, ovšem každá má specifickou sadu funkcí a vlastností. Záleží tedy pouze na konkrétním scénáři použití. Pro naše další testovací účely byla vybrána databáze InfluxDB. Nevyžaduje žádné závislosti ani složitou hardwarovou infrastrukturu, může běžet bez problému na jedné serverové instanci, je schopná ukládat data dlouhodobě při zachování původní přesnosti a poskytuje uspokojivou množinu dotazovacích funkcí.

	Druid	**InfluxDB**	**KairosDB**	**OpenTSDB**
Verze	0.90	0.10	1.1.1	2.1
Licence	Apache 2.0	MIT	Apache 2.0	LGPLv2.1+
Uložiště	v paměti	zabudované	Cassandra	HBase
Jazyk	Java	Go	Java	Java
Přesnost	ms	ns	ms	ms
Schéma databáze	Ano	Ne	Ne	Ne
Vkládání dat	HTTP	HTTP, UDP, CLI	HTTP, Telnet	HTTP, CLI
Dotazovací jazyk	JSON	SQL	JSON	JSON
Komprese	Ano	Ano	Ano	Ano

Tabulka 5: Srovnání databázových systémů pro ukládání časových řad

[20] http://www.gnuplot.info/
[21] https://github.com/Ticketmaster/metrilyx-2.0
[22] https://github.com/box/StatusWolf
[23] https://github.com/facebookarchive/tsdash
[24] https://bosun.org/

5 Nasazení InfluxDB a PostgreSQL

Tato kapitola se zabývá přípravou a nasazením databázových systémů InfluxDB a PostgreSQL. Nejprve je vysvětlen scénář, jímž se budeme v další části textu zabývat. Dále následuje popis funkcí, které jsou od systému očekávány, návrh způsobu uložení dat a popis systémů, na kterých databázové systémy poběží.

Pro experimenty s databázemi byla vybrána dopravní data. Jedná se o smyšlený scénář inspirovaný reálnými daty, který byl navržen s cílem otestování možností databázových systémů pro ukládání časových řad. Modelová situace vypadá následovně. Je zapotřebí ukládat a dotazovat dopravní data ze silnic a dálnic v České republice. Data jsou sbírána v minutových intervalech z 22000 senzorů. Bude se pracovat s daty za tři měsíce – konkrétně se bude jednat o první kvartál roku 2015. Při frekvenci vkládání 22000 záznamů za minutu, obdrží systém přes 31 miliónů záznamů denně, tedy necelé 3 miliardy záznamů za kvartál. Data se budou především zapisovat a číst. Jednou vložená data už se nebudou aktualizovat. Úkolem bude porovnat, jak si s testy dokáže poradit kromě databáze InfluxDB i volně dostupná relační databáze PostgreSQL.

Pro testování byla vybrána data společnosti HERE, která poskytuje veřejné REST API s různými reálnými dopravními daty. K dispozici jsou 4 druhy dat:

- Traffic Incident Data – agregované informace o událostech v dopravě,
- Traffic Map Tile Overlays – předpřipravené mapové dlaždice,
- Traffic Flow Data – data z dopravy v reálném čase,
- Traffic Flow Availability – obecné informace o dopravě.

Za využívání API se musí platit, ale existuje i časově omezená varianta zdarma. Pro experimenty bylo potřeba vygenerovat data inspirovaná právě výše zmíněnými daty z dopravy v reálném čase. Místo reálných dat ze služby HERE byla vytvořena data umělá. Jejich popis se nachází v Tabulce 6. Umělé datové kolekce dopravních dat byly vytvářené pomocí generátoru Petra Zubka (2016) [18]. Struktura generovaných dat odpovídá dopravním datům ze služby společnosti HERE. Nejedná se o nepravidelně generovaná data. Generátor se snaží napodobit reálný provoz a data vytvářet podle přesně stanovených pravidel. Snaží se simulovat plynulý provoz a dokonce i občasnou zácpu či havárii. Pro účel našeho experimentu jsou data dostačující.

Jméno atributu	Popis	Datový typ
CREATED_TIMESTAMP	Datum vytvoření souboru	unsigned int
DE	Textový popis cesty	string
PC	Traffic Message Channel (TMC) kód	string
LI	Unikátní textový identifikátor řady	string
FF	Rychlost – volný proud	short
UT	Doba platnosti predikce	short
SU	Současná rychlost	short
SN	Doba změny dat	int
JF	Ukazatel zácpy	int
TY	Typ lokace	short
TABLE_ID	TMC tabulka	int
SP	Normální rychlost	int
PFSP	Predikovaná rychlost	int
LE	Délka položky	int
EBU_COUNTRY_CODE	TMC kód země	short
CN	Spolehlivost měření	short
FC	Počet křivek cesty	int

Tabulka 6: Datový slovník [43]

5.1 Funkční analýza

Při výběru funkcí pro testování byla věnována pozornost tomu, které funkce by mohl fiktivní systém pracující s dopravními daty využívat. Jedná se o funkcionalitu společnou pro oba databázové systémy. Funkce byly vybrány tak, aby otestovaly, jakým způsobem se databáze vypořádají s různými typy dotazů. Jsou zde zahrnuty bodové dotazy, rozsahové dotazy, agregační funkce, seskupování záznamů a třídění. Funkcím byla pro účely testování přidělena četnost, se kterou by je fiktivní systém mohl využívat. Odhadovaná četnost operací je uvedena v Tabulce 7. Pro testování byly vybrány tyto operace:

1. všechny informace ze senzoru v konkrétním čase,
2. současná rychlost ze senzoru v konkrétním čase,
3. dotaz vracející TMC kód, typ lokace, ukazatel zácpy a současnou rychlost ze zadaného senzoru za hodinu,
4. dotaz vracející TMC kód, typ lokace, ukazatel zácpy a současnou rychlost ze tří senzorů za hodinu,

5. dotaz vracející TMC kód, typ lokace, ukazatel zácpy a současnou rychlost ze zadaného senzoru za den,
6. dotaz vracející TMC kód, typ lokace, ukazatel zácpy a současnou rychlost ze tří senzorů za den,
7. dotaz vracející TMC kód, typ lokace, ukazatel zácpy a současnou rychlost ze zadaného senzoru za hodinu v konkrétním typu lokace,
8. celkový součet ukazatelů zácpy v konkrétním čase,
9. počet měření v konkrétním čase,
10. průměrná rychlost na všech úsecích v zadaném čase,
11. rozdíl mezi minimální rychlostí ze všech úseků a maximální rychlostí ze všech úseků v konkrétním čase,
12. nejvyšší rychlost ze všech úseků v konkrétním čase,
13. nejnižší rychlost ze všech úseků v konkrétním čase,
14. osmdesátiprocentní percentil rychlosti v zadaný den,
15. unikátní typy lokace v zadaný den,
16. nejvyšší tři rychlosti podle unikátního kódu řady v konkrétním čase,
17. nejnižší tři rychlosti podle unikátního kódu řady v konkrétním čase,
18. směrodatná odchylka ze všech úseků v konkrétním čase,
19. nejvyšší rychlost z jednoho senzoru za hodinu,
20. průměrná rychlost z jednoho senzoru za den,
21. průměrná rychlost z jednoho senzoru za měsíc,
22. nejvyšší rychlost z 20 senzorů za hodinu podle senzoru,
23. percentil rychlosti z 20 senzorů za den,
24. průměrná rychlost z 20 senzorů za měsíc podle senzoru,
25. vkládání dat – jednou za minutu 22000 záznamů.

Funkce 1-24 budeme v dalším textu označovat zkratkami Q1–Q24 a operaci vkládání budeme označovat zkratkou I1.

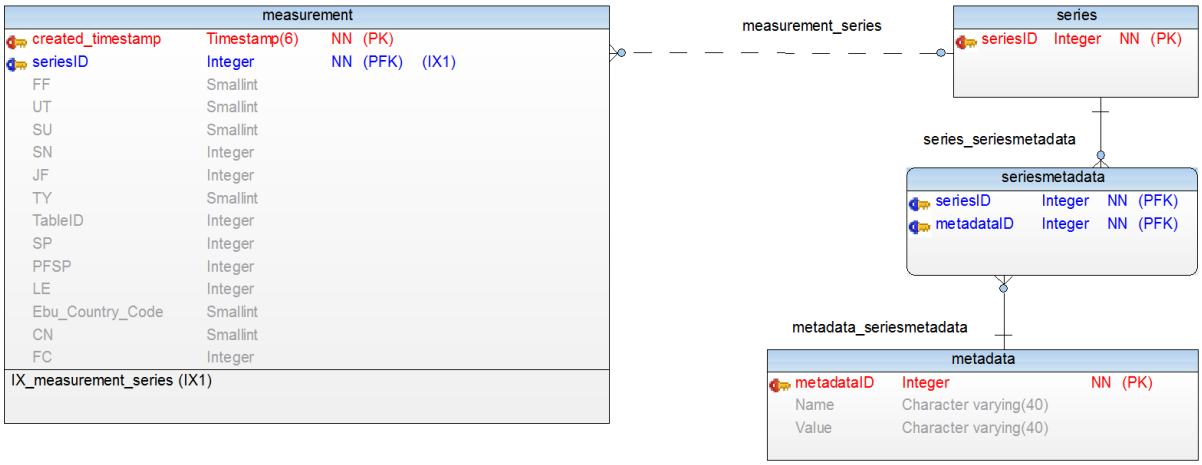
Zkratka operace	Odhadovaná četnost za hodinu
Q1	120
Q2	120
Q3	90
Q4	90
Q5	30
Q6	30
Q7	30
Q8	2
Q9	2
Q10	2
Q11	2
Q12	2
Q13	2
Q14	2
Q15	2
Q16	2
Q17	2
Q18	2
Q19	120
Q20	30
Q21	1
Q22	120
Q23	30
Q24	1
I1	60

Tabulka 7: Odhadované četnosti operací za hodinu

5.2 Relační model

InfluxDB je bez schématu, není tedy nutné vytvářet model. Pro uložení dopravních dat v databázovém systému PostgreSQL byl navržen relační model, který je vidět na Obrázku 5. Vychází ze způsobů uložení časových řad v relačních databázích popisovaných v kapitole 2. Konkrétně vychází ze způsobu, který navrhli Lee a Elmasri (1998) [10]. Tento způsob uložení se hodí, když se ukládají data se stejnou časovou granularitou [4]. Všechny záznamy jsou uloženy v jedné tabulce nazvané *measurement*. Metadata s informacemi o časových řadách se nacházejí v samo-

statné tabulce *metadata*. Návrh porušuje třetí normální formu vzhledem k tomu, že se datum v tabulce opakuje a vzniká tak redundance, což může časem vést ke zpomalení výkonu systému.



Obrázek 5: Relační model pro PostgreSQL

5.3 Fyzický návrh

V této části bude popsáno, jak se postupovalo při tvorbě fyzického návrhu u InfluxDB a PostgreSQL. U PostgreSQL byly vyzkoušeny čtyři varianty fyzického návrhu a z nich vybrána ta nejvhodnější.

5.3.1 InfluxDB

InfluxDB neumožňuje ovlivňovat způsob fyzického uložení dat a vytvářet vlastní indexy na libovolné atributy. Volbou schématu lze však jistými způsoby ovlivnit dotazování a to i v negativním směru. Data lze snadno uložit i špatným způsobem a je třeba si dát pozor na to, co chceme s daty provádět za operace. Autoři uvádějí v dokumentaci několik vzorových situací, kterým se při ukládání dat vyhnout [44]. Pokud se má nějaký atribut používat ve WHERE podmínce a pokud nenabývá příliš mnoha hodnot je vhodné použít jej jako tag. Tagy jsou totiž indexovány. Všechny hodnoty jsou uloženy do jednoho měření *traffic*. Textové atributy *DE*, *PC* a *LI* jsou uloženy jako tagy. Tagy by neměly obsahovat hodnoty s vysokou kardinalitou, to ale není tento případ. Zbylé hodnoty jsou uloženy jako pole (fields). Návrh schématu pro tento scénář byl konzultován přímo s autory InfluxDB a je tedy ověřené, že je korektní. InfluxDB neumožňuje ovlivnit nastavení komprese – jen nepřímo pomocí změny v konfiguračním souboru. V něm se dá změnit maximální počet bodů ukládaných v blocích datového souboru, což může zlepšit kompresi, ale zpomalit dotazování [53].

5.3.2 PostgreSQL

PostgreSQL nabízí více možností ladění fyzického návrhu než InfluxDB. Fyzický návrh se ladí změnou použitých datových struktur, vytvořením dalšího indexu atp. Návrh se bude ladit s ohledem na tento konkrétní scénář uložení dopravních dat a sadu dotazů specifikovanou ve funkční analýze. Bude vyzkoušeno několik možností fyzického návrhu. Všechny budou otestovány a porovnány a nejvhodnější varianta se v dalších experimentech použije pro srovnání s InfluxDB. Data se budou především vkládat a číst. Jednou zapsaná data by se již neměla aktualizovat ani mazat. Je tedy vhodné použít jako typ tabulky haldu a pro indexování B-strom [9]. PostgreSQL ostatně používá pouze tabulky typu halda, ale záznamy je možné shlukovat i podle indexu [16]. Kromě B-stromu nabízí PostgreSQL i další speciální typy indexu, které se dají využít při fyzickém návrhu. Seznam indexů v PostgreSQL:

- B-strom – univerzální index, nejčastěji používaný,
- GiST – Generalized Search Tree (GiST) [12] je optimalizovaný pro fulltextové vyhledávání a speciální datové typy včetně TIMESTAMP,
- GIN – Generalized Inverted Index (GIN) [12] je určen pro fulltextové vyhledávání a je rychlejší než GiST, ale pomaleji se aktualizuje,
- SP-GiST - Generalized Search Tree [12],
- hash – starší než GiST a GIN, které jsou dnes výkonnější než hash [12],
- btree_gist – hodí se pro vytváření složených indexů různých datových typů [12],
- btree_gin – spojuje výhody B-stromu a GIN [12].

Základní fyzický návrh, ze kterého se bude při ladění vycházet, je vidět v Tabulce 8.

Tabulka 8: Fyzický návrh pro PostgreSQL

Název	**Tabulka**	**Sloupec**	**Typ**
Measurement	Measurement	-	heap table
MeasurementKey	Measurement	created_timestamp, seriesID	B-strom primární klíč
Measurement_Series	Measurement	seriesID	cizí klíč
Series	Series	-	heap table
SeriesKey	Series	seriesID	B-strom primární klíč
SeriesMetadata	SeriesMetadata	-	heap table
SeriesMetadataKey	SeriesMetadata	seriesID, metadataID	B-strom primární klíč
Series_SeriesMeta	SeriesMetadata	seriesID	cizí klíč
Meta_SeriesMeta	SeriesMetadata	metadataID	cizí klíč
Metadata	Metadata	-	heap table
MetadataKey	Metadata	metadataID	B-strom primární klíč

5.3.2.1 Ladění fyzického návrhu V návrhu jsou zohledněna následující fakta: filtrování se bude provádět nejčastěji podle času a unikátního identifikátoru řady, specifikovaná sada dotazů má vysokou selektivitu a většina dotazů se ptá na rychlost.

Je možné vytvořit složený index nad časem publikování a unikátním identifikátorem řady. Výhoda složeného indexu má oproti více samostatným indexům je v tom, že sám pokryje více dotazů, nemusí se pak vytvářet více samostatných indexů, čímž se sníží celková velikost indexu. Složené indexy umožňují vytvářet pouze B-strom, GiST a GIN. B-strom je jako jediný schopen zaručit unikátnost záznamů, proto je také použit jako složený primární klíč. Nelze tedy použít jen jeden index, je nutno využít kombinaci více indexů. Index typu GIN na sloupec *created_timestamp* použít nelze, protože potřebuje datový typ TIMESTAMP s časovým pásmem (timezone). Sloupec *created_timestamp* má datový typ TIMESTAMP bez timezone, jelikož se pracuje pouze s daty pro jednu zemi. Jak bylo zmíněno v kapitole 2, časové řady se často indexují pomocí prostorových indexů typu R-strom. Dřívější verze PostgreSQL umožňovaly použití R-stromů, ale později byla tato metoda odstraněna, protože neměla žádné výhody oproti indexu GiST, který R-strom v aktuální verzi nahrazuje. Jako jedna z variant se tedy použije složený index typu GiST. GiST umožňuje indexovat i datový typ TIMESTAMP podobným způsobem jako B-strom. Požadován je však složený index GiST skládající se z času a identifikátoru řady. Toho lze dosáhnout pomocí speciálního indexu btree_gist – používá se když je třeba vytvořit složený index, který se skládá ze sloupců indexovatelných pomocí GiST, ale některé ze sloupců jsou i běžné datové typy. GiST bohužel neumožňuje seřazení dat podle atributu. Odzkouší se tedy ještě varianta složeného B-strom indexu setříděného podle času sestupně.

V případě, že by se pracovalo s větším množstvím dat, bylo by vhodné nastavit pro tabulku *measurement* data partitioning a rozdělit data do více podtabulek spadajících pod jednu rodičovskou. Mohlo by se tím dosáhnout vyšších rychlostí dotazů. Velikost intervalu pro rozdělení dat je třeba experimentálně stanovit, aby byla optimální vzhledem k dotazování a vkládání dat. Mohlo by se jednat např. o měsíc, čtvrtletí, půl roku atp.

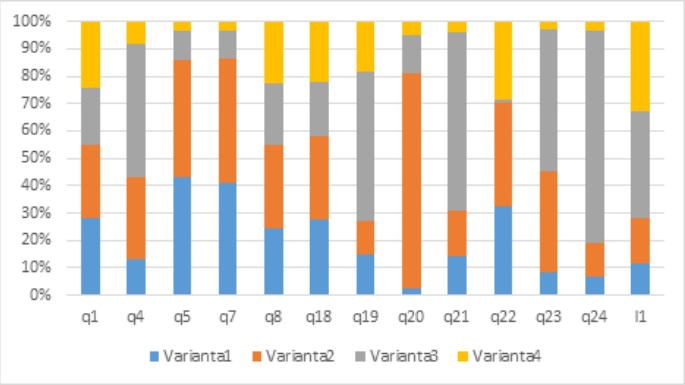
Co se možností komprese týče, v relačních databázích se často na tabulku typu halda používá bloková komprese a prefixová komprese u B-stromu. Tyto možnosti však v PotsgreSQL chybí. Jedinou možností je komprimovat řetězce při zaplnění šířky řádku, což probíhá automaticky. Byla by zde ještě možnost data uložit do zkomprimovaného souboru a na soubor v tabulce odkazovat, tato možnost ovšem v tomto případě není vhodná – výrazně by se tím ztížilo dotazování.

Byly vyzkoušeny čtyři navrhované varianty (včetně výše zmíněné základní varianty) optimalizace fyzického návrhu:

1. Index jen na primární klíč (publicationTime, seriesID).
2. Index na primární klíč (publicationTime, seriesID) a cizí klíč (seriesID)
3. Index na primární klíč (publicationTime, seriesID) a cizí klíč (seriesID) a speciální typ idexu pro datum GiST index (publicationTime, seriesID)

4. Index na primární klíč (publicationTime, seriesID) a cizí klíč (seriesID) a index na (seriesID, publicationTime DESC)

Na Obrázku 6 jsou vidět procentuální výsledky rychlostí vykonání operací pro jednotlivé fyzické návrhy. Čím menší část grafu varianta zabírá tím lépe. Výsledky byly u některých typů dotazů přibližně stejné, proto jsou zobrazeny pouze vybrané operace. V tabulce 9 jsou srovnány jednotlivé varianty z pohledu využití místa na disku. Ačkoliv to na první pohled nemusí být zřejmé, nejlépe dopadla čtvrtá varianta, protože u často vykonávaných dotazů měla nejkratší časy. Agregační dotazy na všech řadách zvládala nejlépe třetí varianta, ale u často vyskytovaných dotazů se jí tolik nedařilo a navíc je na tom nejhůře z pohledu využití místa na disku a rychlosti vkládání. Indexování pomocí GiST trvá totiž poměrně dlouho a index je až dvojnásobně větší oproti klasickému B-stromu.



Obrázek 6: Souhrnné srovnání variant fyzického návrhu podle rychlosti operací

	Varianta1	**Varianta2**	**Varianta3**	**Varianta4**
Data (GB)	224	224	224	224
Index (GB)	96	162	368	241
Celkem (GB)	320	386	592	465

Tabulka 9: Srovnání fyzického návrhu podle místa na disku

5.4 Testovací prostředí

Pro testování byly vytvořeny dva virtuální servery. Windows Server 2012, na kterém běžel PostgreSQL a CentOS 6.7 pro InfluxDB. Jak je možné vidět v tabulce 19 CentOS využívá 500GB pevný disk. Protože PostgreSQL neumožňuje nastavení komprese musela být serveru udělena vyšší disková kapacita. Windows Server měl tedy 2 disky jeden 500GB a druhý 600GB. Jako procesory byly použité procesory Intel s parametry – Intel Xeon CPU E5-2695 v3 2.30GHz.

Operační systém	**Disk**	**Paměť**	**Procesor**
CentOS 6.7	500GB	8GB RAM	2 CPU
CentOS 6.7	500GB	12GB RAM	3 CPU
Windows Server 2012 R2	1,1TB	8GB RAM	2 CPU
Windows Server 2012 R2	1,1TB	12GB RAM	3 CPU

Tabulka 10: Hardware testovacího operačního systému

6 Experimenty nad databází

Kapitola se zabývá přípravou prostředí pro experimenty, popisem aplikací použitých pro vykonání experimentů a měření vytížení systémových prostředků a popisem samotných experimentů a způsobu jejich vykonávání.

6.1 Příprava prostředí pro experimenty

Pro zajištění co možná nejpřesnějších výsledků bylo třeba nachystat prostředí k vykonávání experimentů a patřičně nakonfigurovat databázové systémy. Proto byly vypnuty všechny procesy, jež by zbytečně zatěžovaly systémové prostředky a jež nebyly nezbytné pro chod operačního systému. Z disku byly vymazány nadbytečné soubory, aby pro experimenty bylo k dispozici co nejvíce místa na disku.

6.1.1 Konfigurace InfluxDB

Použitý operační systém CentOS byl na virtuální server nainstalován v minimální verzi bez uživatelského rozhraní. Práce s ním probíhala pouze přes příkazovou řádku. Na server byl nainstalován databázový systém InfluxDB verze 0.10. Pro spuštění byl místo defaultního konfiguračního souboru vytvořen nový. V konfiguračním souboru bylo ke zrychlení dotazování vypnuto ověřování hesla a odesílání anonymních statistik. Na serveru se musely otevřít příslušné porty pro komunikaci s InfluxDB tzn. v *iptables* se musela vytvořit nová pravidla pro povolení TCP portů 8086 a 8083.

6.1.2 Konfigurace PostgreSQL

Na Windows Server byl nainstalován databázový systém PostgreSQL ve verzi 9.4. PostgreSQL, bylo zapotřebí nakonfigurovat pro daný hardware provedením změn v konfiguračním souboru *postgresql.conf*. Pro pomoc s konfigurací byla využita online kalkulačka PgTune[25]. Bylo třeba nastavit hodnoty těchto parametrů: max_connections, shared_buffers, effective_cache_size, work_mem, maintenance_work_mem, checkpoint_segments, wal_buffers, default_statistics_target a pro logování statistik vykonávání databázových operací se musel nastavit log_statement na hodnotu *all*. Data vyžadovala 224 GB volného místa na disku, k tomu bylo třeba připočítat velikost indexu a adresáře pro ukládání dočasných souborů. Z tohoto důvodu byl databázový adresář přemístěn ze systémového disku C (500 GB) na větší disk E (600GB). Na disku E nebyl dostatek místa pro vytvoření všech indexů, proto budou ukládány v separátním uložišti na systémovém disku, kde byl pro ně vytvořen samostatný tablespace. Pro co nejrychlejší naplnění databáze byly odstraněny indexy a integritní omezení. Heslo pro přístup k databázi bylo uloženo v globálních proměnných operačního systému, jinak by se při vkládání dat přes terminál psql muselo ručně zadávat a proces vkládání by nemohl probíhat automaticky.

[25] http://pgtune.leopard.in.ua/

6.1.3 Naplnění daty

Data pro naplnění databází se generovala přes generátor od Petra Zubka[18]. Databáze měly být naplněny daty za první kvartál roku 2015. Velikost dat vygenerovaných za dobu tří měsíců pro vložení do InfluxDB by přesahovala 1TB. Namísto vygenerování všech dat a následného vložení byl generátor dat použit takovým způsobem, aby se data generovala ve formátu odpovídajícím dané databázi a rovnou se po vygenerování na databázový server odesílala. Databáze InfluxDB byla plněna dávkami o velikosti 5000 záznamů (což je doporučovaná velikost dávky v dokumentaci InfluxDB). K naplnění databáze bylo použito příkazu *COPY* posílaného přes aplikaci *psql*[26]. PostgreSQL bylo plněno dávkami o velikosti cca 1GB po dnech. Po naplnění databáze se musely obnovit integritní omezení a indexy a data se musela ještě zaindexovat.

6.2 Sledování vytížení systémových prostředků

Hlavním kritériem pro porovnání databázových systémů byl u experimentů čas vykonávání operace na straně serveru. Kromě času se monitorovalo i vytížení procesoru, paměti a přístupy na disk. Pro hodnot systémových prostředků byly využity nástroje doporučované na stránkách PostgreSQL v sekci analýza výkonu[52]. Měřilo se především vytížení procesoru, paměti a disku.

6.2.1 Sledování vytížení prostředků u PostgreSQL

Na Windows byl pro záznam sledovaných ukazatelů použit nástroj *perfmon* (Performance Monitor)[27]. Perfmon je monitorovací nástroj s univerzálním využitím. Umožňuje sledovat a zaznamenávat výkonnostní ukazatele, události a konfigurační informace. Výkonnostních ukazatelů je k dispozici celá řada, my jsme však použili jen malou podmnožinu z nich, která bude porovnatelná s ukazateli měřenými v linuxovém systému. Do logu v CSV formátu byly zaznamenávány tyto hodnoty:

- (PDH-CSV 4.0) (Central Europe Standard Time)(-60) – čas,
- Disk Transfers/sec – počet čtení a zápisů za sekundu,
- Disk Bytes/sec – počet bytů přečtených a zapsaných bytů za sekundu,
- % Processor Time – procento času procesoru stráveného vykonáváním vlákna procesu,
- Working Set – množství naalokované paměti procesu.

[26] http://www.postgresql.org/docs/9.4/static/app-psql.html
[27] https://technet.microsoft.com/en-us/library/cc749249.aspx

6.2.2 Sledování vytížení prostředků u InfluxDB

K měření vytížení systémových prostředků na operačním systému CentOS byla využita kombinace více nástrojů současně. Jednalo se o nástroje iostat[28], pidstat[29], iotop[30] a vmstat[31]. Tyto nástroje do logů v pravidelných intervalech zaznamenávaly měřené hodnoty.

Příkaz **iostat** slouží k monitorování vstupně/výstupní zátěže zařízení. Pozoruje dobu, kdy jsou zařízení aktivní ve vztahu k průměrným dobám přenosu. Díky přehledům generovaným tímto příkazem, je možné patřičně změnit systémovou konfiguraci a vybalancovat tak zátěž fyzických disků. Zde jsou pro nás zajímavé hodnoty:

- %user – počet procent využití času procesoru během vykonávání na úrovni uživatele,
- %system – počet procent využití času procesoru během vykonávání na úrovni systému,
- %iowait – počet procent využití času procesoru během vykonávání na úrovni systému,
- %idle – procento času, kdy procesor nebyl vytížen,
- r/s – počet požadavků na čtení obdržených za sekundu,
- w/s – počet požadavků na zápis obdržených za sekundu.

Příkaz **pidstat** je používán k monitorování samostatných procesů. Na standardní výstup vypisuje všechny aktivity procesu specifikovaného pomocí příznaku -p nebo pro každý proces řízený jádrem Linuxu za pomocí příznaku -ALL. V přehledu jsou zobrazeny pouze procesy, které mají nějaké nenulové statistické hodnoty.

Zde jsou pro nás zajímavé hodnoty:

- %CPU – celkový počet procent využití času procesoru,
- %MEM – množství fyzické paměti používané procesem,
- VSZ – množství virtuální paměti používané procesem.

Přehledy zobrazované příkazem **vmstat** ukazují informace o procesech, paměti, stránkování, blokovém vstupu a výstupu a aktivitách procesoru. První zobrazená data zobrazí průměry od posledního restartu. Další přehledy zobrazují informace ze zadaného časového intervalu.

Zde jsou pro nás zajímavé hodnoty:

- %free – množství volné paměti
- %swpd – množství využívané virtuální paměti

[28] http://linux.die.net/man/1/iostat
[29] http://linux.die.net/man/8/vmstat
[30] http://linux.die.net/man/1/iotop
[31] http://linux.die.net/man/1/pidstat

- %cache – množství paměti využívané jako cache

Příkaz **iotop** sleduje využití vstupů a výstupů jádrem systému Linux a zobrazuje tabulku vytížení vstupů a výstupů procesů nebo vláken systému. Je třeba mít povoleny systémové hodnoty CONFIG_TASK_DELAY_ACCT a CONFIG_TASK_IO_ACCOUNTING.

Zde jsou pro nás zajímavé hodnoty:

- DISK READ,
- DISK WRITE,
- IO – celkové I/O procesu,
- SWAPIN – swapovací aktivita procesu.

6.3 Implementace pomocných aplikací

Vzhledem k tomu, že experimenty probíhaly opakovaně, dávalo smysl automatizovat testovací proces. V rámci práce bylo naimplementováno několik pomocných aplikací v C# a C++ pro usnadnění práce s testováním a vyhodnocováním výsledků. Jedná se o tyto aplikace:

1. aplikace pracující s generátorem dat,
2. testovací aplikace pro zasílání dotazů na databázi a testování souběžného zápisu a čtení,
3. aplikace na vkládání dat do databáze,
4. aplikace na parsování logů z databázových systémů a monitorovacích nástrojů (perfmon, pidstat atp.).

První aplikace je napsaná v C++ a používá knihovnu pro generování dat od Petra Zubka[18]. Aplikace generuje data, vkládá je do databáze a spouští monitorování a logování změn. Celý proces probíhá automaticky. Pomocí knihovny jsou vygenerována data, následně jsou naformátovány pro zvolenou databázi a vloženy do databázového systému. Při opakovaném použití generátoru vždy narůstalo množství alokované paměti což vedlo ke kolapsu aplikace. Tento problém byl vyřešen tím, že po použití generátoru a uložení dat za jeden den byla činnost aplikace ukončena a vzápětí byl generátor znovu spuštěn – dokud nebyl proces generování a vkládání dat zcela ukončen. Data jsou do databáze vkládána ihned po vygenerování dat za jeden den. Vkládání probíhá u PostgreSQL pomocí terminálu *psql* a u InfluxDB přes CLI programu *curl*[32]. Pomocí programu *plink*[33] se na linuxový server odesílá sada příkazů, jenž má zjišťovat přírůstek místa na disku a spouštět/vypínat aplikace pro monitorování vytížení systémových prostředků. U PostgreSQL se přírůstek místa na disku po vložení dat zjišťuje odesláním dotazu na databázi přes terminál pgsql.

[32]https://curl.haxx.se/
[33]http://www.chiark.greenend.org.uk/ sgtatham/putty/download.html

Druhá aplikace slouží k testování dotazování, k aktualizaci a k souběžnému zápisu a čtení dat. Aplikace je napsaná v C#. Při spuštění je inicializován náhodný generátor vždy stejnou hodnotou, aby testování probíhalo pro každou variantu testovacího scénáře stejně. Část kódu byla použitelná pro obě databáze, ale většina se musela přizpůsobit danému databázovému systému. Pro komunikaci s PostgreSQL se používá knihovna Npgsql[34]. Zasílání requestů na HTTP API InfluxDB umožňuje třída *HttpWebRequest*. Pro správu monitorování systémových prostředků a zjišťování velikosti dat na disku, se stejně jako u první aplikace používá *plink* u InfluxDB a *psql* u PostgreSQL. HttpWebRequesty posílané na InfluxDB občas zhavarovaly na vypršení času spojení, což bylo způsobeno pomalým vykonáváním dotazů na serveru. Bylo nutné navýšit čas vypršení spojení (TimeOut). Testování souběžného zápisu a čtení probíhá ve vláknech. Pro zápis dat je vyhrazeno samostatné vlákno, které jednou za minutu posílá data na databázi. Pro posílání dotazů je vyhrazeno deset vláken. V náhodně určených intervalech posílají na databázi dotazy v pevně daném maximálním množství za minutu.

Třetí aplikace se používala během testovacího scénáře vkládání dat. Aplikace pouze odesílá předem vygenerovaná statická data na databázové servery.

Aplikace na parsování logů má pomoci se získáním výsledků z logů obou databází a monitorovacích nástrojů perfmon, pidstat, iostat, iotop a vmstat. Každý log má svůj specifický formát. Aplikace zadaný log převede do CSV formátu, který se dá dále zpracovat např. pomocí programu Microsoft Excel.

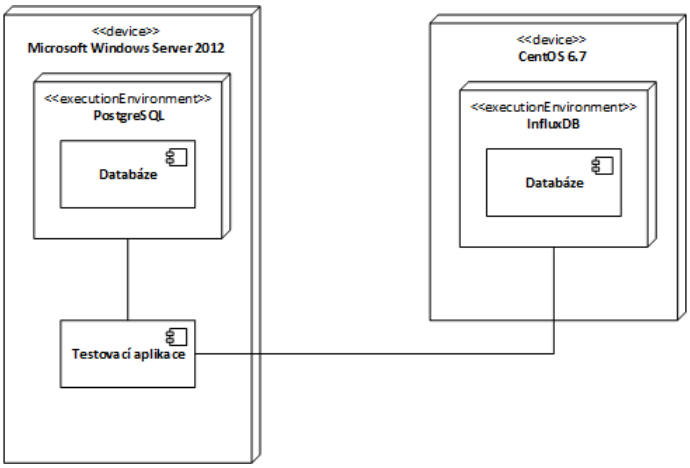
6.4 Testovací scénáře

V této části jsou popsány prováděné experimenty a všechny jejich varianty. Testy byly prováděny s různými parametry a konfigurací hardware. Na počátku testování obsahovaly databáze vždy stejná data za tři měsíce. Pro objektivní výsledky měření bylo důležité dodržet několik kroků. Bylo třeba zajistit, aby na serveru kromě software nutného pro běh operačního systému a vykonání experimentů neběžela žádná jiná aplikace. Po provedení experimentu je zapotřebí uvést databázový systém zpět do původního stavu. To u PostgreSQL znamenalo obnovit původní data, pročistit databázové statistiky a pomocí příkazu VACUUM aktualizovat statistiky plánovače a uvolnit nepoužívané místo na disku (např. data označená jako smazaná). V některých případech bylo nutné celé schéma smazat a data vložit znovu. U InfluxDB se pro každý test použila kopie databáze v původním stavu. Role v rámci testu vypadaly následovně:

- databázový systém – běží na serveru a zaznamenává informace o operacích do svého interního logu,
- logovací nástroj – běží na serveru a monitoruje vytížení systémových prostředků (perfmon, pidstat atd.),
- testovací aplikace – zasílá požadavky na databázový server.

[34] http://www.npgsql.org/

Způsob vykonání testů je přiblížen na obrázku 7. Během testování InfluxDB bude CentOS sloužit jako server a Windows Server jako klient, jenž zasílal na server příkazy. V rámci testování PostgreSQL sloužil Windows Server zároveň jako server i klient. Testovací aplikace pouze zasílá příkazy/dotazy na databázi a v paměti zabírala jen cca 10 MB. Na výsledky měření by to nemělo mít zásadní dopad. Vytížení prostředků operačního systému bylo navíc měřeno přímo pro daný proces.



Obrázek 7: Diagram nasazení

6.4.1 Vkládání dat

Parametry testu jsou popsány v Tabulce 11. Pro test byla předem vygenerována data za týden pomocí generátoru z práce Petra Zubka[18]. Data jsou rozdělena do souborů podle velikosti vkládané dávky. V dokumentaci InfluxDB je uvedeno, že nezvládne příliš velké dávky dat najednou. Doporučují se dávky velikosti 5000 záznamů, proto byly testované velikosti dávek zvoleny v okolí tohoto čísla.

Tabulka 11: Testovací scénář: vkládání dat

Inicializace	V databázi jsou uloženy tři měsíce měření.
Popis testování	Do databáze budou vkládána data z dalšího týdne po určitém počtu záznamů.
Sledované parametry	Rychlost vložení dat Přírůstek na disku Sledování prostředků: vytížení disku, paměti, cpu.
Varianty testu	1. Konfigurace HW: a. CentOS 500GB 8GB Ram, 2 CPU b. CentOS 500GB 12GB Ram, 3 CPU c. Windows 1,1TB 8GB RAM 2 CPU d. Windows 1,1TB 12GB RAM 3 CPU
	2. Databázový systém a. InfluxDB b. PostgreSQL
	3. Počet vkládaných záznamů najednou a. 1000 b. 5000 c. 10000

6.4.2 Dotazování dat

Parametry testu jsou popsány v Tabulce 12. Testují se všechny typy dotazů z funkční analýzy Q1-Q24. Dotazy jsou sekvenčně po jednom zasílány na databázi a každý z nich je proveden tisíckrát. Testovací aplikace vždy inicializuje generátor náhodných hodnot stejnou hodnotou, dotazy tak budou směřovat v každé variantě testu do stejných části datového souboru.

Tabulka 12: Testovací scénář: dotazování dat

Inicializace	V databázi jsou uloženy tři měsíce měření.
Popis testování	Na databázi se budou s určitým počtem opakování posílat dotazy stejného typu s náhodně vygenerovanými parametry. Budou se testovat všechny typy dotazů z předem definované množiny.
Sledované parametry	Rychlost vykonání dotazu Sledování prostředků: vytížení disku, paměti, cpu.
Varianty testu	1. Konfigurace HW: 1. CentOS 500GB 8GB Ram, 2 CPU 2. CentOS 500GB 12GB Ram, 3 CPU 3. Windows 1,1TB 8GB RAM 2 CPU 4. Windows 1,1TB 12GB RAM 3 CPU
	2. Databázový systém a. InfluxDB b. PostgreSQL
Počet provedení jednoho typu dotazu	1000

6.4.3 Souběžné čtení a zápis

Test má simulovat hodinu běžného provozu databáze. Každou minutu budou vkládána nová data a na databázi budou průběžně v náhodně zvolených intervalech zasílány dotazy z výše zmíněné množiny dotazů. Jedná se o dotazy, které se v běžném provozu volají několikrát za minutu. Dotazy, které by se v běžném provozu volaly jednou za měsíc nebo za den byly vynechány. Používají se dotazy: Q1, Q2, Q3, Q8, Q10, Q19 a Q22. Záměrně byly vybrány i typy dotazů, se kterými mohou mít databáze problémy. Podrobnější informace o parametrech testu se nacházejí v Tabulce 13. PostgreSQL pro udržení konzistence dat používá interně Multiversion Concurrency Control (MVCC) tzn. každý dotaz vidí databázi v takovém stavu v jakém byla před určitou dobou, bez ohledu na současný stav. Hlavní výhodou MVCC modelu pro správu souběhu je, že MVCC zámky pro dotazování nejsou v konfliktu se zámky pro zápis dat. Čtení tedy nikdy neblokuje zápis a naopak.

Tabulka 13: Testovací scénář: souběžné čtení a zápis

Inicializace	V databázi jsou uloženy tři měsíce měření.
Popis testování	Do databáze budou hodinu vkládána data v množství a intervalu odpovídajícím reálnému provozu Zároveň budou v náhodných intervalech na databázi zasílány dotazy.
Sledované parametry	Rychlost vkládání Rychlost vykonání dotazu Sledování prostředků: vytížení disku, paměti, cpu.
Varianty testu	1. Konfigurace HW: a. CentOS 500GB 8GB Ram, 2 CPU b. CentOS 500GB 12GB Ram, 3 CPU c. Windows 1,1TB 8GB RAM 2 CPU d. Windows 1,1TB 12GB RAM 3 CPU
	2. SŘBD a. InfluxDB b. PostgreSQL
Počet vkládaných záznamů za minutu	22000
Počet dotazů za minutu	1-20

6.4.4 Aktualizace dat

InfluxDB neumožňuje aktualizovat data. V dřívějších verzích to možné bylo, ale po změnách ve způsobu uložení dat, byla tato funkce odebrána. Záznam lze ovšem přepsat záznamem se stejným časem a stejnou množinou tagů. Lze očekávat, že rychlost aktualizace vložením záznamu se bude podobat rychlosti vkládání. Rozdíl může být v tom, že při běžném vkládání přidáváme data nová a ta se většinou umisťují na konec datového bloku. V tomto testu budeme aktualizovat náhodně zvolené záznamy vybrané z celého datasetu. Více informací o testu lze nalézt v Tabulce 14.

Tabulka 14: Testovací scénář: aktualizace dat

Inicializace	V databázi jsou uloženy tři měsíce měření.
Popis testování	V DB budou aktualizovány záznamy v náhodně zvolených záznamech z celého datasetu.
Sledované parametry	Rychlost aktualizace Sledování prostředků: vytížení disku, paměti, cpu.
Varianty testu	1. Konfigurace HW: a. CentOS 500GB 8GB Ram, 2 CPU b. CentOS 500GB 12GB Ram, 3 CPU c. Windows 1,1TB 8GB RAM 2 CPU d. Windows 1,1TB 12GB RAM 3 CPU
	2. Databázový systém a. InfluxDB b. PostgreSQL
	Počet aktualizovaných záznamů a. 1 a. 1000 b. 5000 c. 10000

6.4.5 Mazání dat

InfluxDB nepočítá s tím, že by se data měla mazat (s výjimkou případů, kdy se smažou všechna). Umožňuje smazat buď celou řadu anebo všechna data z celého měření. Srovnání je tedy možné provést jen z pohledu doby mazání celé řady. Testovat dobu smazání celé databáze nemá smysl, protože to není rozhodující faktor při výběru databázového systému. K vykonání testu nebylo nutné psát samostatnou aplikaci. Mazání bylo prováděno zadáním příkazu do konzole. Více informací o testu lze nalézt v Tabulce 15.

Tabulka 15: Testovací scénář: mazání dat

Inicializace	V databázi jsou uloženy tři měsíce měření.
Popis testování	Z databáze bude smazána množina dat z jednoho senzoru.
Sledované parametry	Rychlost mazání Sledování prostředků: vytížení disku, paměti, cpu.
Varianty testu	1. Konfigurace HW: a. CentOS 500GB 8GB Ram, 2 CPU b. CentOS 500GB 12GB Ram, 3 CPU c. Windows 1,1TB 8GB RAM 2 CPU d. Windows 1,1TB 12GB RAM 3 CPU
	2. Databázový systém a. InfluxDB b. PostgreSQL
Počet smazaných záznamů	129 600

7 Vyhodnocení experimentů nad databází

V této kapitole jsou nejprve popsány výsledky jednotlivých testovacích scénářů uvedených v části 6.4. Na konec je provedeno závěrečné zhodnocení obou databázových systémů.

7.1 Výsledky vkládání dat

Jak je možné vidět na Obrázku 8, rychlost vkládání dat byla ve všech případech lepší u InfluxDB než u PostgreSQL. Týden dat se do PostgreSQL ve variantě s výkonnější hardwarovou konfigurací vkládal asi sedm hodin, do InfluxDB něco přes dvě hodiny. Propustnost operací za sekundu je u InfluxDB 26 600 záznamů u PostgreSQL asi 8200 záznamů. Rychlost vkládání byla u InfluxDB ve variantě s lepší hardwarovou konfiurací 3krát vyšší než u PostgreSQL, ve variantě se slabší hardwarovou konfigurací 1,73-krát vyšší než u PostgreSQL.

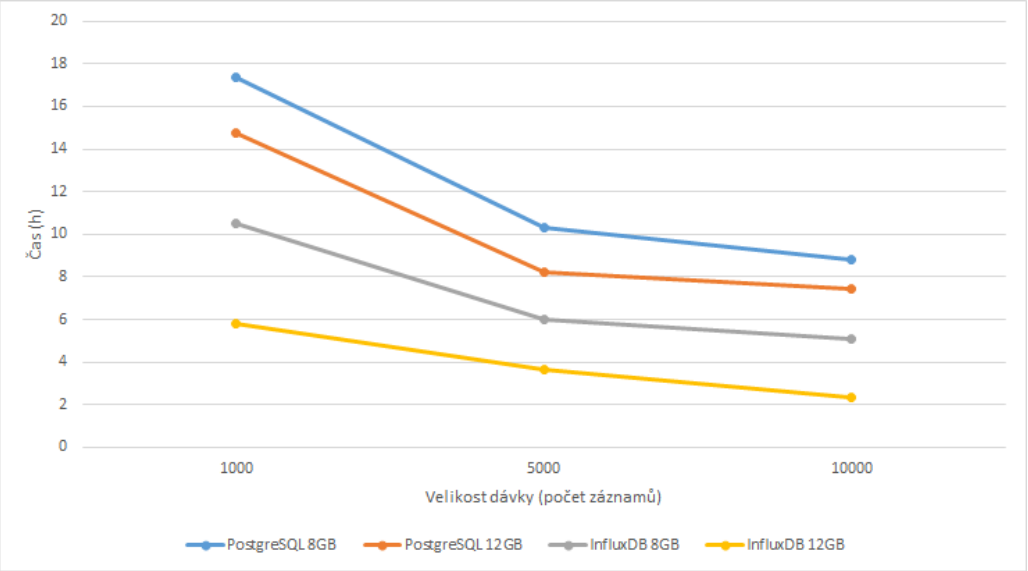
PostgreSQL zatěžoval během vkládání dat procesor v průměru na 10,3 %, nároky na paměť RAM byly minimální – okolo 2%. Průměrný počet dat zapsaných procesem za sekundu byl 4,2 MB/s. Databáze InfluxDB při vkládání zatěžovala CPU z 22,5 %, zabírala 49,9 % paměti a na disk zapisovala v průměru 11,77 MB/s. U InfluxDB byl průměrný přírůstek za den 433 MB. Data se, však průběžně stlačovala (viz architektura databáze 4.2.1), čímž se zlepšovala komprese. Ve výsledku vycházelo, že jeden den (3,1 miliónů záznamů) zabírá v průměru 365 MB. U PostgreSQL byl průměrný přírůstek za den 5,2 GB. Celkem za týden cca 36,4 GB. Ačkoliv bylo u PostgreSQL vkládání pomalejší, obě varianty by splnily požadavky definované testovacím scénářem pro ukládání dopravních dat. InfluxDB je však rozhodně úspornější z hlediska obsazení místa na disku. Jak je možné vidět v Tabulce 16, data za kvartál v InfluxDB zabírají na disku 13krát méně místa než v PostgreSQL. Výsledek není až tak překvapivý vzhledem k tomu, že u PostgreSQL nebyla použita žádná komprese a InfluxDB všechna data automaticky komprimuje.

	InfluxDB	**PostgreSQL**
Celkem (GB)	33	465

Tabulka 16: Srovnání PostgreSQL a InfluxDB podle místa na disku za kvartál

7.2 Výsledky dotazování

Souhrnné srovnání rychlostí dotazů je možné vidět na Obrázku 9, všechny naměřené hodnoty jsou uvedeny v Tabulce 17. InfluxDB je v některých typech dotazů mnohem rychlejší než PostgreSQL. Jedná se o dotazy Q3–Q7 a Q19–Q24. Na druhou stranu, PostgreSQL zase zvládá lépe dotazy Q1–Q2 a Q8–Q18. Agregační dotazy, které u InfluxDB řeší pomaleji, mají společné to, že nefiltrují výsledky podle metadat v podmínce WHERE. Metadata jsou totiž indexována a když se v dotazu neuvedou musí databáze procházet a dekódovat všechny bloky dat. Vzhledem k tomu, že se celý dataset nevejde do paměti, musí procházet data po částech. PostgreSQL zase u pomalých dotazů Q3–Q7 a Q19–Q24 tráví spoustu času operací Bitmap Heap Scan. Zatímco



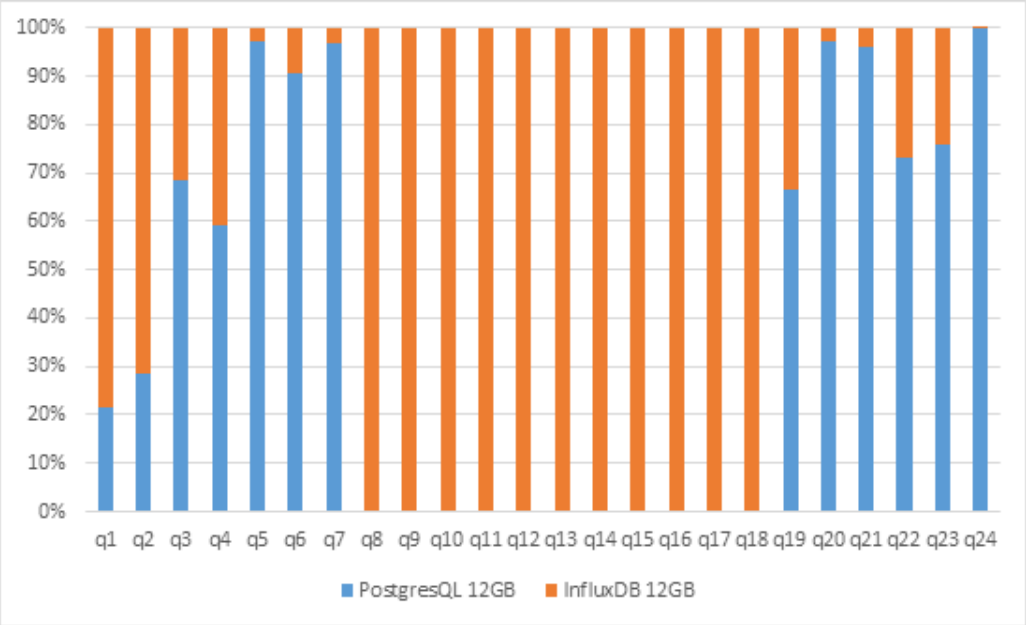
Obrázek 8: Srovnání rychlostí vkládání dat za týden u PostgreSQL a InfluxDB

u agregačního dotazu na hodně různých řad databázi stačí provést Index Scan na primárním klíči, zde musí nejprve operace Bitmap Index Scan určit, které stránky by mohly obsahovat požadovaná data, a poté je sekvenčně projde pomocí operace Bitmap Heap Scan. Bitmap Index Scan se používá proto, že se ptáme na mnoho řádků tabulky, ale ne na všechny.

Agregační dotazy Q8–Q18 u InfluxDB podle měření vyžadují velké množství RAM. Jaký vliv na rychlost dotazů mělo navýšení kapacity RAM a přidání CPU na serverech je vidět na Obrázcích 10 a 11. Z Obrázku 10 lze vyčíst, že výkonnější hardwarová konfigurace pomohla InfluxDB ke zrychlení pomalých dotazů Q1, Q2 a Q8–Q18. Dotazy se zrychlily v některých případech skoro až o polovinu. Zbylé dotazy navýšení výkonu příliš neovlivnilo a vykonávaly se přibližně stejnou rychlostí. Graf na obrázku srovnává situaci u PostgreSQL. Došlo k mírnému zrychlení některých dotazů, např. Q1, Q2, Q21 a Q23 o cca 10%. PostgreSQL nemá tak vysoké nároky na paměť při dotazování jako InfluxDB a je pro něj dostačující i méně výkonná hardwarová konfigurace.

7.3 Výsledky souběžného čtení a zápisu

PostgreSQL test zvládl bez problému. Zvládal vkládat data i odpovídat na 20 dotazů za minutu. InfluxDB zvládl pouze cca 17–18 dotazů za minutu, což je však dáno tím, že pod náporem nedokázal odpovídat na dva typy dotazů – Q8 a Q10. Vytížení procesoru se během testu u obou databázových systémů pohybovalo kolem 4,5 %. Paměť InfluxDB však byla v průměru neustále vytížena na 89 % , kvůli agregačním dotazům Q8 a Q10. Kvůli nim totiž musela databáze procházet všechny bloky dat, protože neobsahují indexovaná metadata ve WHERE podmínce. Výsledky testu je možné vidět v Tabulce 18. Z výsledků je možné vyčíst, jak experiment ovlivnila změna hardware. U InfluxDB více paměti pomohlo zrychlit dotazy Q1, Q2, Q3, Q19 a Q22. Dotaz



Obrázek 9: Procentuální srovnání rychlostí dotazů u PostgreSQL a InfluxDB

Q10 se zrychlil, ale dotaz Q8 se zase zpomalil. Tato databáze má s tímto typem dotazu zcela zřejmě problém. U PostgreSQL se navýšení hardware příliš neprojevilo. Některé dotazy byly rychlejší jiné zase pomalejší, rozdíl tvořil většinou jen pár desítek milisekund. Když výsledky porovnáme s výsledky v Tabulce 17, můžeme si všimnout, že průměrná rychlost operací byla v tomto testu někdy dokonce až 5krát pomalejší. Důvodem je, že se na databázi posílalo více dotazů současně a proto déle trvalo než je databáze všechny obsloužila.

Tabulka 18: Srovnání rychlostí operací během souběžného čtení a zápisu

	InfluxDB 8 GB	InfluxDB 12 GB	PosgreSQL 8 GB	PosgreSQL 12 GB
q1	614	81	145	83
q2	561	120	195	83
q3	2 232	488	752	864
q8	954 034	3 248 455	789	862
q10	1 438 828	308 184	818	835
q19	2 285	477	739	862
q22	5 066	609	3 615	3 642
insert1	3 025	412	8 204	7 685

7.4 Výsledky aktualizace dat

Podle výsledků z Tabulky 19 můžeme vidět, že aktualizace záznamů je rychlejší u InfluxDB. Nevýhodou je, že se musí celý záznam vložit znovu. Pokud se jedná o změnu v nedávno uložených

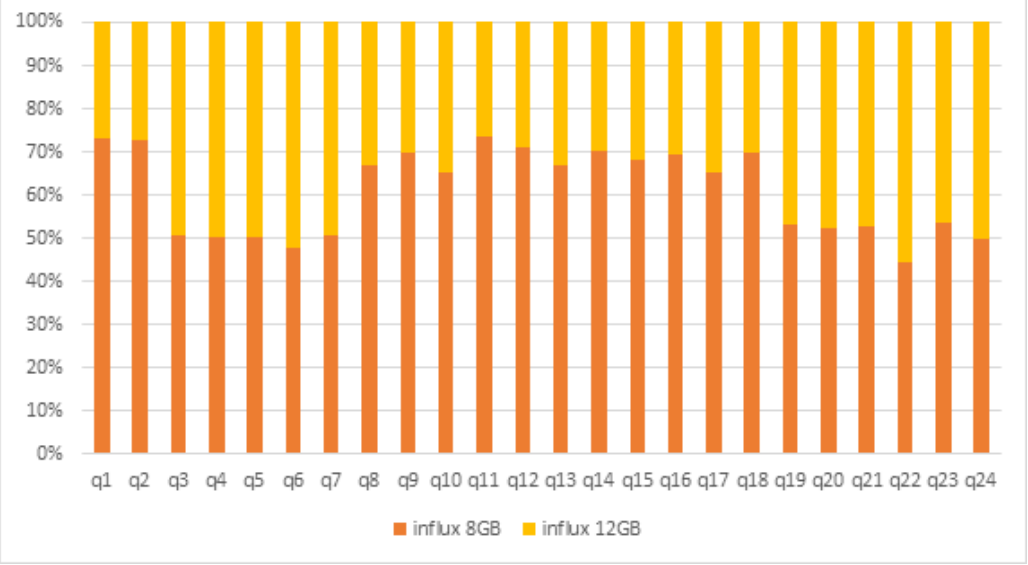
Tabulka 17: Srovnání průměrných rychlostí dotazů (v milisekundách)

	PostgreSQL 8 GB	PostgreSQL 12 GB	InfluxDB 8 GB	InfluxDB 12 GB
q1	30	18	179	65
q2	28	24	160	60
q3	459	487	228	224
q4	444	391	273	269
q5	7 861	8 335	245	242
q6	3 597	3 222	305	332
q7	7 608	7 510	269	262
q8	429	452	559 267	277 145
q9	415	411	709 647	305 810
q10	452	437	536 984	288 199
q11	436	436	727 213	259 715
q12	482	437	717 522	291 193
q13	414	434	645 993	321 085
q14	308	370	709 769	302 594
q15	486	460	623 938	289 611
q16	427	467	647 002	283 225
q17	384	438	652 985	347 927
q18	438	428	594 730	257 663
q19	481	441	252	224
q20	8 728	7 496	245	224
q21	37 511	29 546	1 398	1 263
q22	2 239	2 150	638	793
q23	18 411	13 996	5 189	4 501
q24	475 623	430 135	666	667

datech, tak je to změna poměrně nenáročná, ale pokud se zapisuje do starších dat musí se znovu zapsat celý datový soubor. Při aktualizaci se u PostgreSQL provádí Index Scan primárního klíče a podle něj se nalezne umístění aktualizovaného řádku v tabulce. Během vykonávání operace využívala databáze PostgreSQL v průměru 3,2% výkonu CPU. Průměrný počet dat zpracovaných procesem byl 2,4 MB/s. Vytížení operační paměti se pohybovalo mezi 1-2%. Aktualizace v InfluxDB zatížila CPU z 3% a proces zabíral v průměru 70% operační paměti. Průměrný počet dat zapsaných na disk byl pouze 4 KB/s.

	PostgreSQL1	**PostgreSQL2**	**InfluxDB1**	**InfluxDB2**
1	0.16	0.31	0.089	0.0022
1000	21.14	18.66	0.8	0.05
5000	72.49	73.73	2.9	0.12
10000	144.65	129.48	5.8	1.9

Tabulka 19: Srovnání rychlostí aktualizace záznamů (v milisekundách)



Obrázek 10: Procentuální srovnání rychlostí u InfluxDB s rozdílným hardware

7.5 Výsledky mazání dat

Operace mazání je u InfluxDB pomalejší, ale u dat typu časových řad to tolik nevadí, protože se většinou nemažou vůbec. Výsledky je možné vidět v tabulce 20.

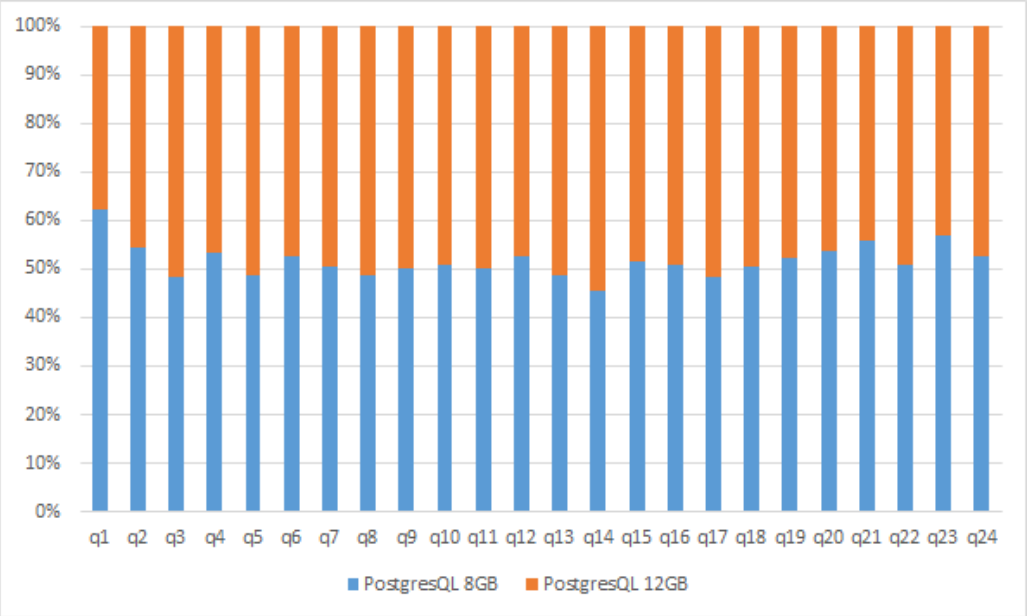
	PostgreSQL 8GB	**PostgreSQL 12GB**	**InfluxDB 8GB**	**InfluxDB 12GB**
Čas (s)	182	151	3354	422

Tabulka 20: Srovnání rychlostí mazání dat

V průběhu mazání dat byl procesor u PostgreSQL vytížen v průměru na 16,7%, množství naalokované operační paměti bylo cca 2% a disk byl vytížen čtením i zápisem dohromady na 31 MB/s. U InfluxDB byl procesor vytížen na 26,5%, operační paměť na 98.6% a počet dat čtených z disku byl 72 MB/s a zapisovaných 53 MB/s. Při mazání řady v PostgreSQL se provádí v tabulce *measurement* Bitmap Index Scan indexu *ix_series_fk(series_id)* a následně Bitmap Heap Scan a mazání příslušných záznamů. Operace mazání v InfluxDB probíhá ve dvou fázích, nejprve se operace zaznamená do Write Ahead Logu, poté se vyčistí paměť cache od mazaných dat. Databáze si poté v paměti udržuje záznam o smazaných datech tzv. *tombstone*. V druhé fázi při provádění změn z Write Ahead Logu se provede nejprve operace delete a přepíšou se všechny datové soubory obsahující smazaná data. Po úspěšném dokončení mazání se odstraní označení *tombstone* z paměti.

7.6 Shrnutí

PostgreSQL má oproti InfluxDB minimální nároky na paměť a procesor, ale zase vyžaduje 13krát větší diskovou kapacitu. InfluxDB umí rychleji vkládat a aktualizovat data. PostgreSQL zase



Obrázek 11: Procentuální srovnání rychlostí u PostgreSQL s rozdílným hardware

zvládá rychleji operaci delete. Co se dotazování týče, záleží na typu dotazu. InfluxDB mnohem lépe zvládá dotazy na časové řady a umí pracovat s velkými objemy dat. Pro testovací scénář ukládání dopravních dat v podobě časových řad se databázový systém InfluxDB ukázal jako vhodný kandidát.

8 Závěr

Provedené experimenty ukázaly přednosti i slabé stránky srovnávaných databázových technologií. Potvrdily uváděné vlastnosti specializovaných databázi pro ukládání časových řad, jejich optimalizaci pro ukládání velkých objemů dat, kompresi dat, rychlost vyřízení určitých dotazů i úsporu času s laděním fyzického návrhu. Rovněž se ukázalo, které typy dotazů nejsou pro tyto specializované technologie vhodné. PostgreSQL například dosáhl lepších výsledků u agregačních dotazů na všechny řady v konkrétním časovém okamžiku. Z experimentů rovněž vyplynula vhodnost uplatnění databází specializovaných pro ukládání časových řad k ukládání a vyhodnocování velkého objemu dopravních dat.

Zpracování diplomové práce mi rozšířilo obzory v oblasti databází specializovaných na časové řady. Odkrylo množství dostupných produktů, založených na různých architekturách a různých způsobech vkládání dat a jejich uložení. V návaznosti na tuto diplomovou práci by bylo zajímavé porovnání i s jinými relačními databázemi typu Oracle, Microsft SQL Server nebo IBM Informix, které poskytují i jiné datové struktury pro fyzické uložení dat než PostgreSQL.

Jakub Vašica

Literatura

[1] ANDLINGER, Paul. Time Series DBMS as a new trend? In: *DB-Engines* [online]. 2015 [cit. 2016-04-18]. Dostupné z: http://db-engines.com/en/blog_post//45

[2] ANDRÉ-JÖNSSON, Henrik. *Indexing Strategies For Time Series Data*. Linköping, Sweden, 2002. ISBN 91-7373-346-6.

[3] ASSENT, Ira, Ralph KRIEGER, Farzad AFSCHARI a Thomas SEIDL. *The TS-Tree: Efficient Time Series Search and Retrieval* [online]. Nantes, France, 2008 [cit. 2016-04-17]. Dostupné z: https://www.researchgate.net/publication/221103762_The_TS-tree_efficient_time_series_search_and_retrieval

[4] CASTILLEJOS, Abel. *Management of Time Series Data* [online]. Canberra, Australia, 2006 [cit. 2016-04-17]. Dostupné z: http://www.canberra.edu.au/researchrepository/file/82315cf7-7446-fcf2-6115-b94fbd7599c6/1/full_text.pdf

[5] DUNNING, Ted a Ellen FRIEDMAN. *Time Series Databases: New Ways to Store and Access Data* [online]. 2014. O'Reilly Media, 2014 [cit. 2016-04-16]. ISBN 978-1-4919-14702-2. Dostupné z: http://info.mapr.com/rs/mapr/images/Time_Series_Databases.pdf

[6] HAN, Jiawei a Micheline KAMBER *Data Mining: Concepts and Techniques* 2nd ed., Morgan Kaufmann, 2006, Chapter 8: http://web.engr.illinois.edu/~hanj/cs512/bk2chaps/chapter_8.pdf

[7] HANČLOVÁ, Jana a Lubor TVRDÝ. *Úvod do analýzy časových řad* [online]. VŠB-TU Ostrava, 2003 [cit. 2016-04-16]. Dostupné z: http://gis.vsb.cz/pan-old/Skoleni_Texty/TextySkoleni/AnalyzaCasRad.pdf

[8] CHINDA, Kishore a Ravi VIJAY. Get started now with the Informix TimeSeries solution. In: *IBM* [online]. 2012 [cit. 2016-04-18]. Dostupné z: http://www.ibm.com/developerworks/data/library/techarticle/dm-1203timeseries/index.html

[9] KEJSER, Thomas. Clustered Index Vs. Heap. In: *Fighting Bad Data Modeling* [online]. 2014 [cit. 2016-04-29]. Dostupné z: http://kejser.org/clustered-indexes-vs-heaps/

[10] Implementations Options for Time-Series Data. LEE, Jae a Ramez ELMASRI. *Temporal databases: research and practice*. New York: Springer, c1998, s. 115-128. ISBN 3540645195.

[11] MOUDRÝ, Martin. *Rozšíření HDF5 o ukládání časových řad*. Ostrava, 2016.

[12] Indexes. OBE, Regina a Leo HSU. *PostgreSQL Up and Running*. Sebastopol: O'Reilly Media, 2012, s. 79-80. ISBN 978-1-449-32633-3.

[13] Time Series, a Neglected Issue in Temporal Database Research? SCHMIDT, Duri, Angelika DITTRICH, Werner DREYER a Robert MARTI. *Recent advances in temporal databases: proceedings of the International Workshop on Temporal Databases, Zurich, 17-18 September 1995*. New York: Springer, c1995, s. 214-232. ISBN 3540199454.

[14] SOROKIN, Anatoly, Gene SELKOV a Igor GORYANIN. A user-defined data type for the storage of time series data allowing efficient similarity screening. *European Journal of Pharmaceutical Sciences*. 2012, extbf46(4), 272-274.

[15] VIET, Huynh a Duong ANH. *M-tree as an Index Structure for Time Series Data* [online]. Ho Chi Minh City, Vietnam, 2013 [cit. 2016-04-17]. Dostupné z: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6482381&tag=1`

[16] WINAND, Markus. Index-Organized Tables and Clustered Indexes. In: *SQL Indexing Tutorial* [online]. 2011 [cit. 2016-04-21]. Dostupné z: `http://use-the-index-luke.com/sql/clustering/index-organized-clustered-index`

[17] YAGLEY, Mike. Optimizing Relational Databases for Time Series Data (Time Series Overview Part 3). In: *TempoIQ* [online]. 2013 [cit. 2016-04-17]. Dostupné z: https://blog.tempoiq.com/blog/2013/04/22/optimizing-relational-databases-for-time-series-data-time-series-database-overview-part-3

[18] ZUBEK, Petr. *Implementace adaptérů pro přístup k datům reprezentovaných časovými řadami*. Ostrava, 2016.

[19] Tsm1 - DESIGN. *Github* [online]. 2015 [cit. 2016-04-23]. Dostupné z: `https://github.com/influxdata/influxdb/blob/master/tsdb/engine/tsm1/DESIGN.md`

[20] Writing Data. *OpenTSDB* [online]. 2016 [cit. 2016-04-23]. Dostupné z: `http://opentsdb.net/docs/build/html/user_guide/writing.html`

[21] Aggregations. *Druid* [online]. 2016 [cit. 2016-04-23]. Dostupné z: `http://druid.io/docs/0.9.0/querying/aggregations.html`

[22] Compression. *DataStax Documentation* [online]. 2016 [cit. 2016-04-29]. Dostupné z: `https://docs.datastax.com/en/datastax_enterprise/4.0/datastax_enterprise/compression.html`

[23] About RRDtool. *Http://oss.oetiker.ch/* [online]. 2014 [cit. 2016-04-18]. Dostupné z: `http://oss.oetiker.ch/rrdtool/index.en.html`

[24] Informix TimeSeries solution architecture. *IBM* [online]. 2015 [cit. 2016-04-18]. Dostupné z: `https://www.ibm.com/support/knowledgecenter/SSGU8G_12.1.0/com.ibm.tms.doc/ids_tms_015.htm?lang=en`

[25] InfluxDB Version 0.10 Documentation. *InfluxData Documentation* [online]. 2016 [cit. 2016-04-18]. Dostupné z: https://docs.influxdata.com/influxdb/v0.10/

[26] *Blueflood DB* [online]. 2013 [cit. 2016-04-18]. Dostupné z: http://blueflood.io/

[27] Storage Engine. *InfluxData - Documentation* [online]. 2015 [cit. 2016-04-23]. Dostupné z: https://docs.influxdata.com/influxdb/v0.9/concepts/storage_engine/

[28] HBase Schema. *OpenTSDB* [online]. 2016 [cit. 2016-04-23]. Dostupné z: http://opentsdb.net/docs/build/html/user_guide/backends/hbase.html

[29] FAQ - OpenTSDB - A Distributed, Scalable Monitoring System. *OpenTSDB* [online]. 2016 [cit. 2016-04-23]. Dostupné z: http://opentsdb.net/faq.html

[30] Cassandra: internal storage. In: *Hadoop Magazine* [online]. 2014 [cit. 2016-04-21]. Dostupné z: http://hadoopmag.com/cassandra-internal-storage/

[31] *Prometheus - Monitoring system time series database* [online]. 2016 [cit. 2016-04-18]. Dostupné z: https://prometheus.io/

[32] Riak TS. *Basho Documentation* [online]. 2011 [cit. 2016-04-18]. Dostupné z: http://docs.basho.com/riak/ts/1.2.0/

[33] Documentation. *Druid Interactiva Analytics at Scale* [online]. 2016 [cit. 2016-04-18]. Dostupné z: http://druid.io/docs/0.9.0/design/index.html

[34] Documentation: What is Druid? *Druid* [online]. 2016 [cit. 2016-04-20]. Dostupné z: http://druid.io/docs/0.9.0/design/design.html

[35] Documentation: Querying *Druid* [online]. 2016 [cit. 2016-04-20]. Dostupné z: http://druid.io/docs/0.9.0/querying/querying.html

[36] Documentation: Ingestion Spec *Druid* [online]. 2016 [cit. 2016-04-20]. Dostupné z: http://druid.io/docs/0.9.0/ingestion/

[37] OpenTSDB - A Distributed, Scalable Monitoring System: How does OpenTSDB work? *OpenTSDB* [online]. 2016 [cit. 2016-03-23]. Dostupné z: http://opentsdb.net/overview.html

[38] Pushing data. *KairosDB* [online]. 2015 [cit. 2016-04-23]. Dostupné z: https://kairosdb.github.io/docs/build/html/PushingData.html

[39] Aggregators. *OpenTSDB* [online]. 2016 [cit. 2016-04-23]. Dostupné z: http://opentsdb.net/docs/build/html/user_guide/query/aggregators.html

[40] Query Metrics. *KairosDB* [online]. 2015 [cit. 2016-04-23]. Dostupné z: https://kairosdb.github.io/docs/build/html/restapi/QueryMetrics.html

[41] Kdb+ time-series data software. *Kx* [online]. 2016 [cit. 2016-04-18]. Dostupné z: https://kx.com/software.php

[42] *KairosDB* [online]. 2015 [cit. 2016-04-18]. Dostupné z: https://kairosdb.github.io/

[43] Traffic API Developer's Guide. *Build applications with HERE Maps API and SDK Platform Access - HERE Developer* [online]. 2016 [cit. 2016-04-27]. Dostupné z: https://developer.here.com/rest-apis/documentation/traffic/topics_v6.1/example-flow.html

[44] Schema Design. *InfluxData Documentation* [online]. 2016 [cit. 2016-04-21]. Dostupné z: https://docs.influxdata.com/influxdb/v0.10/concepts/schema$_a nd_d ata_l ayout$/

[45] The complete list of all time series databases for your IoT project. In: *Erol.si* [online]. 2016 [cit. 2016-04-18]. Dostupné z: http://www.erol.si/2015/01/the-complete-list-of-all-timeseries-databases-for-your-iot-project/

[46] FAQ - Graphite 0.10.0 documentation. *Graphite Documentation* [online]. [cit. 2016-04-18]. Dostupné z: https://graphite.readthedocs.org/en/latest/faq.html

[47] Documentation for OpenTSDB 2.2. *OpenTSDB - A Distributed, Scalable Monitoring System* [online]. 2016 [cit. 2016-04-18]. Dostupné z: http://opentsdb.net/docs/build/html/

[48] Axibase Time Series Database. *Axibase* [online]. 2016 [cit. 2016-04-18]. Dostupné z: http://axibase.com/products/axibase-time-series-database/

[49] Cassandra Schema. *KairosDB* [online]. 2015 [cit. 2016-04-23]. Dostupné z: https://kairosdb.github.io/docs/build/html/CassandraSchema.html

[50] Cisco předpovídá do roku 2019 pětinásobný nárůst objemu dat přenášených po internetu ve střední a východní Evropě a zdvojnásobení průměrné rychlosti připojení. In: *Cisco Systems, Inc* [online]. 2015 [cit. 2016-04-21]. Dostupné z: http://www.cisco.com/c/cs_cz/about/news/2015/20150608.html

[51] Querying or Reading Data. *OpenTSDB* [online]. 2016 [cit. 2016-04-23]. Dostupné z: http://opentsdb.net/docs/build/html/user_guide/query/index.html

[52] Performance Analysis Tools *PostgreSQL Wiki* [online]. 2016 [cit. 2016-04-23]. Dostupné z: https://wiki.postgresql.org/wiki/Performance_Analysis_Tools

[53] Database Configuration. *InfluxData Documentation* [online]. 2016 [cit. 2016-04-21]. Dostupné z: https://docs.influxdata.com/influxdb/v0.10/administration/config/

A Obsah přiloženého DVD

Na přiloženém DVD se nacházejí tyto složky:

- **Text** – obsahuje text bakalářské práce ve formátu pdf,
- **Skripty** – SQL skripty pro operace s databázovými systémy,
- **Aplikace_insert** – zdrojové soubory k aplikaci na vkládání dat,
- **Aplikace_tests** – zdrojové soubory k aplikaci testovací aplikace,
- **Aplikace_generator** – zdrojové soubory k aplikaci na generování dat,
- **Aplikace_log_parser** – zdrojové soubory k aplikaci na získávání dat z logů.